In [1]:
```
# 1. IMPORT LIBRARIES

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from PIL import Image
import cv2
import kagglehub
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_ir
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dr
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score, precision_score, recal
import time
import warnings
warnings.filterwarnings('ignore')

print("✅ Libraries imported successfully!")
print(f"TensorFlow version: {tf.__version__}")
```

```
✅ Libraries imported successfully!
TensorFlow version: 2.19.0
```

In [2]:
```
# ============================================
# 2. LOAD DATASET
# ============================================

print("\n" + "="*60)
print("DOWNLOADING DATASET FROM KAGGLE")
print("="*60)

path = kagglehub.dataset_download("paultimothymooney/chest-xray-pneumonia")
print("✅ Dataset downloaded to:", path)

# Define directories
BASE_DIR = os.path.join(path, 'chest_xray')
TRAIN_DIR = os.path.join(BASE_DIR, 'train')
TEST_DIR = os.path.join(BASE_DIR, 'test')
VAL_DIR = os.path.join(BASE_DIR, 'val')

# Image parameters
IMG_HEIGHT = 150
IMG_WIDTH = 150

def load_images_from_directory(directory, img_height=IMG_HEIGHT, img_width=I
    """Load images from directory with NORMAL and PNEUMONIA subfolders"""
    images = []
    labels = []
    categories = ['NORMAL', 'PNEUMONIA']

    for category_idx, category in enumerate(categories):
        category_path = os.path.join(directory, category)
        if not os.path.exists(category_path):
            print(f"⚠️  Warning: {category_path} does not exist")
            continue

        print(f"Loading {category} images...")
        image_files = os.listdir(category_path)
```

```
        for img_file in image_files:
            if img_file.endswith(('.jpeg', '.jpg', '.png')):
                img_path = os.path.join(category_path, img_file)
                try:
                    img = cv2.imread(img_path)
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    img = cv2.resize(img, (img_width, img_height))
                    images.append(img)
                    labels.append(category_idx)
                except Exception as e:
                    print(f"Error loading {img_path}: {e}")
                    continue

        print(f"✅ Loaded {len([l for l in labels if l == category_idx])} {c

    return np.array(images), np.array(labels)

# Load all data
print("\n" + "="*60)
print("LOADING TRAINING DATA")
print("="*60)
X_train_orig, y_train_orig = load_images_from_directory(TRAIN_DIR)

print("\n" + "="*60)
print("LOADING TEST DATA")
print("="*60)
X_test, y_test = load_images_from_directory(TEST_DIR)

print("\n" + "="*60)
print("LOADING VALIDATION DATA")
print("="*60)
X_val_orig, y_val_orig = load_images_from_directory(VAL_DIR)

print("\n✅ Data loading complete!")
print(f"   Training:   {X_train_orig.shape}")
print(f"   Test:       {X_test.shape}")
print(f"   Validation: {X_val_orig.shape}")

# ============================================
# MERGE ORIGINAL TRAIN + VALIDATION SETS
# ============================================

print("\n" + "="*60)
print("MERGING ORIGINAL TRAIN + VALIDATION SETS")
print("="*60)

X_dev = np.concatenate([X_train_orig, X_val_orig], axis=0)
y_dev = np.concatenate([y_train_orig, y_val_orig], axis=0)

print(f"✅ Combined dataset size: {X_dev.shape[0]} images")

# ============================================
# 3. STRATIFIED 80:20 TRAIN / VALIDATION SPLIT
# ============================================

print("\n" + "="*60)
print("CREATING STRATIFIED 80:20 TRAIN/VAL SPLIT")
print("="*60)

X_train, X_val, y_train, y_val = train_test_split(
    X_dev, y_dev,
    test_size=0.2,
    random_state=42,
```

```
    stratify=y_dev
)

print(f"✅ New splits created:")
print(f"   Training:   {X_train.shape[0]} images")
print(f"   Validation: {X_val.shape[0]} images")
print(f"   Test:       {X_test.shape[0]} images (unchanged)")


# ============================================
# 4. NORMALIZE DATA
# ============================================

print("\n" + "="*60)
print("NORMALIZING PIXEL VALUES")
print("="*60)

X_train_scaled = X_train.astype('float32') / 255.0
X_val_scaled = X_val.astype('float32') / 255.0
X_test_scaled = X_test.astype('float32') / 255.0

print(f"✅ Data normalized to range [{X_train_scaled.min():.2f}, {X_train_s

# ============================================
# 5. SHOW DISTRIBUTION BEFORE BALANCING
# ============================================

print("\n" + "="*60)
print("📊 DATASET DISTRIBUTION (BEFORE BALANCING)")
print("="*60)

print("\nTraining Set:")
print(f"  Total: {len(y_train)} images")
print(f"  NORMAL:    {np.sum(y_train == 0)} ({np.sum(y_train == 0)/len(y_tra
print(f"  PNEUMONIA: {np.sum(y_train == 1)} ({np.sum(y_train == 1)/len(y_tra

print("\nValidation Set:")
print(f"  Total: {len(y_val)} images")
print(f"  NORMAL:    {np.sum(y_val == 0)} ({np.sum(y_val == 0)/len(y_val)*10
print(f"  PNEUMONIA: {np.sum(y_val == 1)} ({np.sum(y_val == 1)/len(y_val)*10

print("\nTest Set:")
print(f"  Total: {len(y_test)} images")
print(f"  NORMAL:    {np.sum(y_test == 0)} ({np.sum(y_test == 0)/len(y_test
print(f"  PNEUMONIA: {np.sum(y_test == 1)} ({np.sum(y_test == 1)/len(y_test

# ============================================
# 6. BALANCE TRAINING SET (50:50)
# ============================================

print("\n" + "="*60)
print("🔧 BALANCING TRAINING SET (50:50 RATIO)")
print("="*60)

# Get indices
train_normal_idx = np.where(y_train == 0)[0]
train_pneumonia_idx = np.where(y_train == 1)[0]

n_normal_train = len(train_normal_idx)
n_pneumonia_train = len(train_pneumonia_idx)

print(f"Before: Normal={n_normal_train}, Pneumonia={n_pneumonia_train}")

# Downsample pneumonia to match normal
```

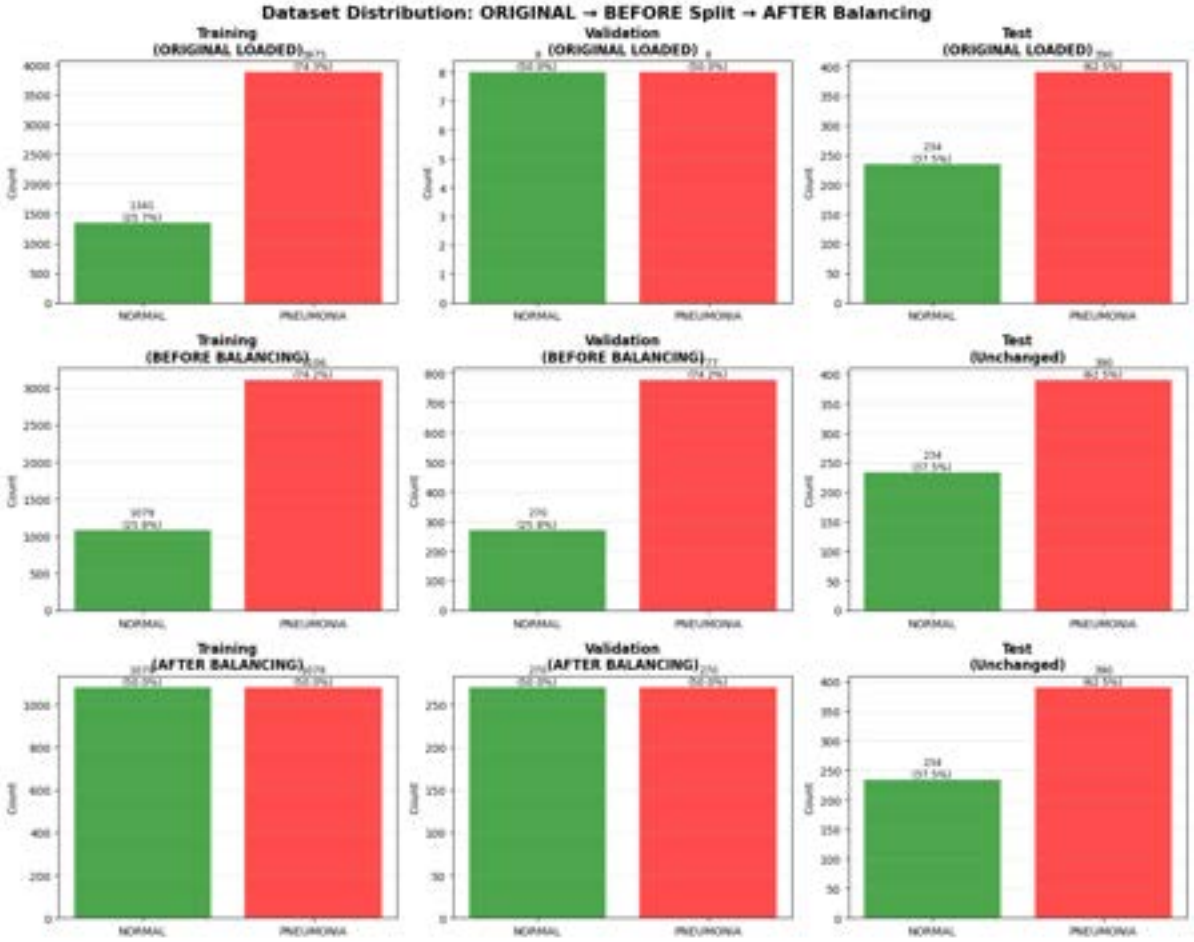
```
np.random.seed(42)
train_pneumonia_downsampled = np.random.choice(train_pneumonia_idx, n_normal

# Combine and shuffle
train_balanced_idx = np.concatenate([train_normal_idx, train_pneumonia_downs
np.random.shuffle(train_balanced_idx)

# Create balanced set
X_train_balanced = X_train_scaled[train_balanced_idx]
y_train_balanced = y_train[train_balanced_idx]

print(f"After:  Normal={np.sum(y_train_balanced == 0)}, Pneumonia={np.sum(y_
print(f"✅ Training set balanced: {len(y_train_balanced)} images (50:50)")

# ============================================
# 7. BALANCE VALIDATION SET (50:50)
# ============================================

print("\n" + "="*60)
print("🔧 BALANCING VALIDATION SET (50:50 RATIO)")
print("="*60)

# Get indices
val_normal_idx = np.where(y_val == 0)[0]
val_pneumonia_idx = np.where(y_val == 1)[0]

n_normal_val = len(val_normal_idx)
n_pneumonia_val = len(val_pneumonia_idx)

print(f"Before: Normal={n_normal_val}, Pneumonia={n_pneumonia_val}")

# Downsample pneumonia to match normal
val_pneumonia_downsampled = np.random.choice(val_pneumonia_idx, n_normal_val

# Combine and shuffle
val_balanced_idx = np.concatenate([val_normal_idx, val_pneumonia_downsampled
np.random.shuffle(val_balanced_idx)

# Create balanced set
X_val_balanced = X_val_scaled[val_balanced_idx]
y_val_balanced = y_val[val_balanced_idx]

print(f"After:  Normal={np.sum(y_val_balanced == 0)}, Pneumonia={np.sum(y_va
print(f"✅ Validation set balanced: {len(y_val_balanced)} images (50:50)")

# ============================================
# 8. FINAL SUMMARY
# ============================================

print("\n" + "="*60)
print("✅ DATA PREPARATION COMPLETE!")
print("="*60)

print("\n📊 Final Balanced Distribution:")
print("\nTraining Set:")
print(f"  Total: {len(y_train_balanced)} images")
print(f"  NORMAL:    {np.sum(y_train_balanced == 0)} ({np.sum(y_train_balanc
print(f"  PNEUMONIA: {np.sum(y_train_balanced == 1)} ({np.sum(y_train_balanc

print("\nValidation Set:")
print(f"  Total: {len(y_val_balanced)} images")
print(f"  NORMAL:    {np.sum(y_val_balanced == 0)} ({np.sum(y_val_balanced =
print(f"  PNEUMONIA: {np.sum(y_val_balanced == 1)} ({np.sum(y_val_balanced =
```

```
print("\nTest Set (UNCHANGED):")
print(f"  Total: {len(y_test)} images")
print(f"  NORMAL:    {np.sum(y_test == 0)} ({np.sum(y_test == 0)/len(y_test)
print(f"  PNEUMONIA: {np.sum(y_test == 1)} ({np.sum(y_test == 1)/len(y_test)

print("\n🎯 Ready for model training!")
print("   Use: X_train_balanced, y_train_balanced")
print("   Use: X_val_balanced, y_val_balanced")
print("   Use: X_test_scaled, y_test")

# ============================================
# VISUALIZATION: ORIGINAL -> BEFORE -> AFTER
# ============================================

print("\n" + "="*60)
print("Creating ORIGINAL -> BEFORE -> AFTER visualizations...")
print("="*60)

fig, axes = plt.subplots(3, 3, figsize=(15, 12))
fig.suptitle('Dataset Distribution: ORIGINAL → BEFORE Split → AFTER Balancin

# ROW 1: ORIGINAL LOADED DATA (before train/val split)
datasets_original = [
    (y_train_orig, 'Training\n(ORIGINAL LOADED)', len(y_train_orig)),
    (y_val_orig, 'Validation\n(ORIGINAL LOADED)', len(y_val_orig)),
    (y_test, 'Test\n(ORIGINAL LOADED)', len(y_test))
]

for idx, (labels, title, total) in enumerate(datasets_original):
    normal = np.sum(labels == 0)
    pneumonia = np.sum(labels == 1)

    bars = axes[0, idx].bar(['NORMAL', 'PNEUMONIA'], [normal, pneumonia],
                            color=['green', 'red'], alpha=0.7)
    axes[0, idx].set_title(title, fontweight='bold', fontsize=12)
    axes[0, idx].set_ylabel('Count', fontsize=10)
    axes[0, idx].grid(axis='y', alpha=0.3)

    # Add labels
    for bar, count in zip(bars, [normal, pneumonia]):
        height = bar.get_height()
        axes[0, idx].text(bar.get_x() + bar.get_width()/2., height,
                          f'{int(count)}\n({count/total*100:.1f}%)',
                          ha='center', va='bottom', fontsize=9)

# ROW 2: BEFORE BALANCING (after 85/15 split)
datasets_before = [
    (y_train, 'Training\n(BEFORE BALANCING)', len(y_train)),
    (y_val, 'Validation\n(BEFORE BALANCING)', len(y_val)),
    (y_test, 'Test\n(Unchanged)', len(y_test))
]

for idx, (labels, title, total) in enumerate(datasets_before):
    normal = np.sum(labels == 0)
    pneumonia = np.sum(labels == 1)

    bars = axes[1, idx].bar(['NORMAL', 'PNEUMONIA'], [normal, pneumonia],
                            color=['green', 'red'], alpha=0.7)
    axes[1, idx].set_title(title, fontweight='bold', fontsize=12)
    axes[1, idx].set_ylabel('Count', fontsize=10)
    axes[1, idx].grid(axis='y', alpha=0.3)

    # Add labels
    for bar, count in zip(bars, [normal, pneumonia]):
```

```
        height = bar.get_height()
        axes[1, idx].text(bar.get_x() + bar.get_width()/2., height,
                          f'{int(count)}\n({count/total*100:.1f}%)',
                          ha='center', va='bottom', fontsize=9)

# ROW 3: AFTER BALANCING (50:50 ratio)
datasets_after = [
    (y_train_balanced, 'Training\n(AFTER BALANCING)', len(y_train_balanced)),
    (y_val_balanced, 'Validation\n(AFTER BALANCING)', len(y_val_balanced)),
    (y_test, 'Test\n(Unchanged)', len(y_test))
]

for idx, (labels, title, total) in enumerate(datasets_after):
    normal = np.sum(labels == 0)
    pneumonia = np.sum(labels == 1)

    bars = axes[2, idx].bar(['NORMAL', 'PNEUMONIA'], [normal, pneumonia],
                            color=['green', 'red'], alpha=0.7)
    axes[2, idx].set_title(title, fontweight='bold', fontsize=12)
    axes[2, idx].set_ylabel('Count', fontsize=10)
    axes[2, idx].grid(axis='y', alpha=0.3)

    # Add labels
    for bar, count in zip(bars, [normal, pneumonia]):
        height = bar.get_height()
        axes[2, idx].text(bar.get_x() + bar.get_width()/2., height,
                          f'{int(count)}\n({count/total*100:.1f}%)',
                          ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()
```

```
============================================================
DOWNLOADING DATASET FROM KAGGLE
============================================================
Using Colab cache for faster access to the 'chest-xray-pneumonia' dataset.
✅ Dataset downloaded to: /kaggle/input/chest-xray-pneumonia

============================================================
LOADING TRAINING DATA
============================================================
Loading NORMAL images...
✅ Loaded 1341 NORMAL images
Loading PNEUMONIA images...
✅ Loaded 3875 PNEUMONIA images

============================================================
LOADING TEST DATA
============================================================
Loading NORMAL images...
✅ Loaded 234 NORMAL images
Loading PNEUMONIA images...
✅ Loaded 390 PNEUMONIA images

============================================================
LOADING VALIDATION DATA
============================================================
Loading NORMAL images...
✅ Loaded 8 NORMAL images
Loading PNEUMONIA images...
✅ Loaded 8 PNEUMONIA images

✅ Data loading complete!
   Training:   (5216, 150, 150, 3)
   Test:       (624, 150, 150, 3)
   Validation: (16, 150, 150, 3)

============================================================
MERGING ORIGINAL TRAIN + VALIDATION SETS
============================================================
✅ Combined dataset size: 5232 images

============================================================
CREATING STRATIFIED 80:20 TRAIN/VAL SPLIT
============================================================
✅ New splits created:
   Training:   4185 images
   Validation: 1047 images
   Test:       624 images (unchanged)

============================================================
NORMALIZING PIXEL VALUES
============================================================
✅ Data normalized to range [0.00, 1.00]

============================================================
📊 DATASET DISTRIBUTION (BEFORE BALANCING)
============================================================

Training Set:
  Total: 4185 images
  NORMAL:    1079 (25.8%)
  PNEUMONIA: 3106 (74.2%)

Validation Set:
  Total: 1047 images
```

```
  NORMAL:    270 (25.8%)
  PNEUMONIA: 777 (74.2%)

Test Set:
  Total: 624 images
  NORMAL:    234 (37.5%)
  PNEUMONIA: 390 (62.5%)

============================================================
🔧 BALANCING TRAINING SET (50:50 RATIO)
============================================================
Before: Normal=1079, Pneumonia=3106
After:  Normal=1079, Pneumonia=1079
✅ Training set balanced: 2158 images (50:50)

============================================================
🔧 BALANCING VALIDATION SET (50:50 RATIO)
============================================================
Before: Normal=270, Pneumonia=777
After:  Normal=270, Pneumonia=270
✅ Validation set balanced: 540 images (50:50)

============================================================
✅ DATA PREPARATION COMPLETE!
============================================================

📊 Final Balanced Distribution:

Training Set:
  Total: 2158 images
  NORMAL:    1079 (50.0%)
  PNEUMONIA: 1079 (50.0%)

Validation Set:
  Total: 540 images
  NORMAL:    270 (50.0%)
  PNEUMONIA: 270 (50.0%)

Test Set (UNCHANGED):
  Total: 624 images
  NORMAL:    234 (37.5%)
  PNEUMONIA: 390 (62.5%)

🎯 Ready for model training!
   Use: X_train_balanced, y_train_balanced
   Use: X_val_balanced, y_val_balanced
   Use: X_test_scaled, y_test

============================================================
Creating ORIGINAL -> BEFORE -> AFTER visualizations...
============================================================
```



In [3]:
```
# Task 2

# ============================================
# Import Additional Libraries for Modeling
# ============================================

from sklearn.metrics import (
    f1_score, classification_report, confusion_matrix,
    accuracy_score, precision_score, recall_score
)
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import (
    Conv2D, MaxPooling2D, Flatten, Dense, Dropout,
    BatchNormalization, GlobalAveragePooling2D
)
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.applications import VGG16, MobileNetV2
import time

print("Additional libraries imported successfully!")
```

```
Additional libraries imported successfully!
```

In [4]:
```
# ============================================
# MODEL 1: SIMPLE CNN (BASELINE)
# Training on BALANCED DATASET
# ============================================

print("\n" + "="*60)
print("MODEL 1: SIMPLE CNN (BASELINE)")
print("="*60)

def create_simple_cnn(input_shape=(150, 150, 3)):
```

```
    """
    Simple CNN architecture - Baseline model
    """
    model = Sequential(name='simple_cnn', layers=[
        # First Convolutional Block
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D(2, 2),

        # Second Convolutional Block
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        # Third Convolutional Block
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),

        # Flatten and Dense Layers
        Flatten(),
        Dense(128, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])

    return model

# Create and compile Model 1
model1 = create_simple_cnn()
model1.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

print("\nModel 1 Architecture:")
model1.summary()

# Display training data info
print("\n" + "="*60)
print("TRAINING DATA INFO")
print("="*60)
print(f"Training on BALANCED dataset:")
print(f"  X_train_balanced: {X_train_balanced.shape}")
print(f"  y_train_balanced: {y_train_balanced.shape}")
print(f"  - Normal: {np.sum(y_train_balanced == 0)} (50%)")
print(f"  - Pneumonia: {np.sum(y_train_balanced == 1)} (50%)")
print(f"\nValidating on BALANCED dataset:")
print(f"  X_val_balanced: {X_val_balanced.shape}")
print(f"  y_val_balanced: {y_val_balanced.shape}")
print(f"  - Normal: {np.sum(y_val_balanced == 0)} (50%)")
print(f"  - Pneumonia: {np.sum(y_val_balanced == 1)} (50%)")

# Train Model 1 on BALANCED data
print("\n" + "="*60)
print("Training Model 1 on BALANCED dataset...")
print("="*60)
start_time = time.time()

history1 = model1.fit(
    X_train_balanced, y_train_balanced,
    validation_data=(X_val_balanced, y_val_balanced),
    epochs=15,
    batch_size=32,
    callbacks=[
        EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=
```

```
        ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_l
    ],
    verbose=1
)

model1_time = time.time() - start_time
print(f"\nModel 1 training completed in {model1_time:.2f} seconds")

# Evaluate Model 1 on TEST SET (unchanged, real-world distribution)
print("\n" + "="*60)
print("Evaluating Model 1 on Test Set...")
print("="*60)
print(f"Testing on ORIGINAL (imbalanced) test set:")
print(f"  X_test_scaled: {X_test_scaled.shape}")
print(f"  - Normal: {np.sum(y_test == 0)} ({np.sum(y_test == 0)/len(y_test)*
print(f"  - Pneumonia: {np.sum(y_test == 1)} ({np.sum(y_test == 1)/len(y_tes

y_pred1_prob = model1.predict(X_test_scaled, verbose=0)
y_pred1 = (y_pred1_prob > 0.5).astype(int).flatten()

# Calculate metrics for both classes
f1_model1 = f1_score(y_test, y_pred1, average=None)  # Per-class F1
f1_weighted_model1 = f1_score(y_test, y_pred1, average='weighted')  # Weight

accuracy_model1 = accuracy_score(y_test, y_pred1)  # Overall accuracy

precision_model1 = precision_score(y_test, y_pred1, average=None)  # Per-cla
precision_weighted_model1 = precision_score(y_test, y_pred1, average='weight

recall_model1 = recall_score(y_test, y_pred1, average=None)  # Per-class rec
recall_weighted_model1 = recall_score(y_test, y_pred1, average='weighted')

print(f"\nModel 1 Results on Test Set:")
print(f"  Overall Accuracy: {accuracy_model1:.4f}")
print(f"\n  Class 0 (NORMAL) Metrics:")
print(f"    Precision: {precision_model1[0]:.4f}")
print(f"    Recall:    {recall_model1[0]:.4f}")
print(f"    F1 Score:  {f1_model1[0]:.4f}")
print(f"\n  Class 1 (PNEUMONIA) Metrics:")
print(f"    Precision: {precision_model1[1]:.4f}")
print(f"    Recall:    {recall_model1[1]:.4f}")
print(f"    F1 Score:  {f1_model1[1]:.4f}")
print(f"\n  Weighted Average:")
print(f"    Precision: {precision_weighted_model1:.4f}")
print(f"    Recall:    {recall_weighted_model1:.4f}")
print(f"    F1 Score:  {f1_weighted_model1:.4f}")

print("\n" + "="*60)
print("MODEL 1 COMPLETED!")
print("="*60)
```

```
============================================================
MODEL 1: SIMPLE CNN (BASELINE)
============================================================

Model 1 Architecture:
Model: "simple_cnn"
```

Layer (type)	Output Shape	Para
conv2d (Conv2D)	(None, 148, 148, 32)	
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18,
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73,
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	
flatten (Flatten)	(None, 36992)	
dense (Dense)	(None, 128)	4,735,
dropout (Dropout)	(None, 128)	
dense_1 (Dense)	(None, 1)	

Total params: 4,828,481 (18.42 MB)
Trainable params: 4,828,481 (18.42 MB)
Non-trainable params: 0 (0.00 B)

```
============================================================
TRAINING DATA INFO
============================================================
Training on BALANCED dataset:
  X_train_balanced: (2158, 150, 150, 3)
  y_train_balanced: (2158,)
  - Normal: 1079 (50%)
  - Pneumonia: 1079 (50%)

Validating on BALANCED dataset:
  X_val_balanced: (540, 150, 150, 3)
  y_val_balanced: (540,)
  - Normal: 270 (50%)
  - Pneumonia: 270 (50%)

============================================================
Training Model 1 on BALANCED dataset...
============================================================
Epoch 1/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 13s 103ms/step - accuracy: 0.6461 - loss: 0.6282
- val_accuracy: 0.9630 - val_loss: 0.1144 - learning_rate: 0.0010
Epoch 2/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9259 - loss: 0.1975 -
val_accuracy: 0.9537 - val_loss: 0.1318 - learning_rate: 0.0010
Epoch 3/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9432 - loss: 0.1418 -
val_accuracy: 0.9648 - val_loss: 0.1104 - learning_rate: 0.0010
Epoch 4/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9567 - loss: 0.1364 -
val_accuracy: 0.9704 - val_loss: 0.0855 - learning_rate: 0.0010
Epoch 5/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9667 - loss: 0.1026 -
val_accuracy: 0.9704 - val_loss: 0.0980 - learning_rate: 0.0010
Epoch 6/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9641 - loss: 0.0929 -
val_accuracy: 0.9722 - val_loss: 0.0825 - learning_rate: 0.0010
Epoch 7/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9688 - loss: 0.0906 -
val_accuracy: 0.9741 - val_loss: 0.0652 - learning_rate: 0.0010
Epoch 8/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9770 - loss: 0.0621 -
val_accuracy: 0.9759 - val_loss: 0.0629 - learning_rate: 0.0010
Epoch 9/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9849 - loss: 0.0446 -
val_accuracy: 0.9722 - val_loss: 0.0768 - learning_rate: 0.0010
Epoch 10/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9710 - loss: 0.0647 -
val_accuracy: 0.9722 - val_loss: 0.0740 - learning_rate: 0.0010
Epoch 11/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9871 - loss: 0.0322 -
val_accuracy: 0.9778 - val_loss: 0.0712 - learning_rate: 5.0000e-04

Model 1 training completed in 32.28 seconds

============================================================
Evaluating Model 1 on Test Set...
============================================================
Testing on ORIGINAL (imbalanced) test set:
  X_test_scaled: (624, 150, 150, 3)
  - Normal: 234 (37.5%)
  - Pneumonia: 390 (62.5%)

Model 1 Results on Test Set:
  Overall Accuracy: 0.7788
```

```
  Class 0 (NORMAL) Metrics:
    Precision: 0.9706
    Recall:    0.4231
    F1 Score:  0.5893

  Class 1 (PNEUMONIA) Metrics:
    Precision: 0.7414
    Recall:    0.9923
    F1 Score:  0.8487

  Weighted Average:
    Precision: 0.8273
    Recall:    0.7788
    F1 Score:  0.7514

============================================================
```

In [23]:

```python
# ============================================
# MODEL 2: ENHANCED RESNET (OPTIMIZED)
# Training on BALANCED DATASET
# ============================================

print("\n" + "="*60)
print("MODEL 2: ENHANCED RESNET")
print("="*60)

# Import additional required layers for ResNet
from tensorflow.keras.layers import Input, Add, Activation
from tensorflow.keras import initializers

def residual_block(x, filters, kernel_size=3, stride=1, name=None):
    """
    Residual block with skip connection - OPTIMIZED VERSION
    """
    # Shortcut connection
    shortcut = x

    # Main path with proper initialization
    fx = Conv2D(filters, kernel_size, strides=stride, padding='same',
                kernel_initializer=initializers.HeNormal(seed=42),
                bias_initializer=initializers.Zeros(),
                name=f'{name}_conv1')(x)
    fx = BatchNormalization(
        gamma_initializer=initializers.Ones(),
        beta_initializer=initializers.Zeros(),
        name=f'{name}_bn1')(fx)
    fx = Activation('relu', name=f'{name}_relu1')(fx)

    fx = Conv2D(filters, kernel_size, strides=1, padding='same',
                kernel_initializer=initializers.HeNormal(seed=42),
                bias_initializer=initializers.Zeros(),
                name=f'{name}_conv2')(fx)
    fx = BatchNormalization(
        gamma_initializer=initializers.Ones(),
        beta_initializer=initializers.Zeros(),
        name=f'{name}_bn2')(fx)

    # Adjust shortcut if dimensions changed
    if stride != 1 or x.shape[-1] != filters:
        shortcut = Conv2D(filters, 1, strides=stride, padding='same',
                          kernel_initializer=initializers.HeNormal(seed=42),
```

```
                          bias_initializer=initializers.Zeros(),
                          name=f'{name}_shortcut')(x)
        shortcut = BatchNormalization(
            gamma_initializer=initializers.Ones(),
            beta_initializer=initializers.Zeros(),
            name=f'{name}_shortcut_bn')(shortcut)

    # Add shortcut to main path
    fx = Add(name=f'{name}_add')([fx, shortcut])
    fx = Activation('relu', name=f'{name}_relu2')(fx)

    return fx

def create_enhanced_resnet(input_shape=(150, 150, 3)):
    """
    Enhanced ResNet architecture - More residual blocks with optimal initia
    """
    inputs = Input(shape=input_shape, name='input')

    # Initial convolution with proper initialization
    x = Conv2D(32, 7, strides=2, padding='same',
               kernel_initializer=initializers.HeNormal(seed=42),
               bias_initializer=initializers.Zeros(),
               name='conv1')(inputs)
    x = BatchNormalization(
        gamma_initializer=initializers.Ones(),
        beta_initializer=initializers.Zeros(),
        name='bn1')(x)
    x = Activation('relu', name='relu1')(x)
    x = MaxPooling2D(3, strides=2, padding='same', name='pool1')(x)

    # Stage 1: 32 filters (2 blocks instead of 1)
    x = residual_block(x, 32, name='block1a')
    x = residual_block(x, 32, name='block1b')

    # Stage 2: 64 filters (2 blocks instead of 1)
    x = residual_block(x, 64, stride=2, name='block2a')
    x = residual_block(x, 64, name='block2b')

    # Stage 3: 128 filters (2 blocks instead of 1)
    x = residual_block(x, 128, stride=2, name='block3a')
    x = residual_block(x, 128, name='block3b')

    # Global pooling and classifier
    x = GlobalAveragePooling2D(name='global_pool')(x)
    x = Dropout(0.5, seed=42, name='dropout')(x)

    # Better final layer initialization
    # Bias initialized to account for balanced dataset (50-50 split)
    outputs = Dense(1, activation='sigmoid',
                    kernel_initializer=initializers.GlorotUniform(seed=42),
                    bias_initializer=initializers.Zeros(),
                    name='output')(x)

    model = Model(inputs=inputs, outputs=outputs, name='enhanced_resnet')

    return model

# Create and compile Model 2 with optimized learning rate
model2 = create_enhanced_resnet()
model2.compile(
    optimizer=Adam(learning_rate=0.0003),
    loss='binary_crossentropy',
    metrics=['accuracy']
```

```
)

print("\nModel 2 Architecture:")
model2.summary()

# Display training data info
print("\n" + "="*60)
print("TRAINING DATA INFO")
print("="*60)
print(f"Training on BALANCED dataset:")
print(f"  X_train_balanced: {X_train_balanced.shape}")
print(f"  y_train_balanced: {y_train_balanced.shape}")
print(f"  - Normal: {np.sum(y_train_balanced == 0)} (50%)")
print(f"  - Pneumonia: {np.sum(y_train_balanced == 1)} (50%)")
print(f"\nValidating on BALANCED dataset:")
print(f"  X_val_balanced: {X_val_balanced.shape}")
print(f"  y_val_balanced: {y_val_balanced.shape}")
print(f"  - Normal: {np.sum(y_val_balanced == 0)} (50%)")
print(f"  - Pneumonia: {np.sum(y_val_balanced == 1)} (50%)")

# Train Model 2 on BALANCED data
print("\n" + "="*60)
print("Training Model 2 on BALANCED dataset...")
print("="*60)
start_time = time.time()

history2 = model2.fit(
    X_train_balanced, y_train_balanced,
    validation_data=(X_val_balanced, y_val_balanced),
    epochs=15,
    batch_size=32,
    callbacks=[
        EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=T
        ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr
    ],
    verbose=1
)

model2_time = time.time() - start_time
print(f"\nModel 2 training completed in {model2_time:.2f} seconds")

# Evaluate Model 2 on TEST SET
print("\n" + "="*60)
print("Evaluating Model 2 on Test Set...")
print("="*60)
print(f"Testing on ORIGINAL (imbalanced) test set:")
print(f"  X_test_scaled: {X_test_scaled.shape}")
print(f"  - Normal: {np.sum(y_test == 0)} ({np.sum(y_test == 0)/len(y_test)*
print(f"  - Pneumonia: {np.sum(y_test == 1)} ({np.sum(y_test == 1)/len(y_tes

y_pred2_prob = model2.predict(X_test_scaled, verbose=0)
y_pred2 = (y_pred2_prob > 0.5).astype(int).flatten()

# Calculate metrics for both classes
f1_model2 = f1_score(y_test, y_pred2, average=None)  # Per-class F1
f1_weighted_model2 = f1_score(y_test, y_pred2, average='weighted')  # Weight

accuracy_model2 = accuracy_score(y_test, y_pred2)  # Overall accuracy

precision_model2 = precision_score(y_test, y_pred2, average=None)  # Per-cla
precision_weighted_model2 = precision_score(y_test, y_pred2, average='weight

recall_model2 = recall_score(y_test, y_pred2, average=None)  # Per-class rec
recall_weighted_model2 = recall_score(y_test, y_pred2, average='weighted')
```

```
print(f"\nModel 2 Results on Test Set:")
print(f"  Overall Accuracy: {accuracy_model2:.4f}")
print(f"\n  Class 0 (NORMAL) Metrics:")
print(f"    Precision: {precision_model2[0]:.4f}")
print(f"    Recall:    {recall_model2[0]:.4f}")
print(f"    F1 Score:  {f1_model2[0]:.4f}")
print(f"\n  Class 1 (PNEUMONIA) Metrics:")
print(f"    Precision: {precision_model2[1]:.4f}")
print(f"    Recall:    {recall_model2[1]:.4f}")
print(f"    F1 Score:  {f1_model2[1]:.4f}")
print(f"\n  Weighted Average:")
print(f"    Precision: {precision_weighted_model2:.4f}")
print(f"    Recall:    {recall_weighted_model2:.4f}")
print(f"    F1 Score:  {f1_weighted_model2:.4f}")

print("\n" + "="*60)
print("MODEL 2 COMPLETED!")
print("="*60)
```

```
============================================================
MODEL 2: ENHANCED RESNET
============================================================

Model 2 Architecture:
Model: "enhanced_resnet"
```

Layer (type)	Output Shape	Param #	Connected to
input (InputLayer)	(None, 150, 150, 3)	0	–
conv1 (Conv2D)	(None, 75, 75, 32)	4,736	input[0][0]
bn1 (BatchNormalizatio…	(None, 75, 75, 32)	128	conv1[0][0]
relu1 (Activation)	(None, 75, 75, 32)	0	bn1[0][0]
pool1 (MaxPooling2D)	(None, 38, 38, 32)	0	relu1[0][0]
block1a_conv1 (Conv2D)	(None, 38, 38, 32)	9,248	pool1[0][0]
block1a_bn1 (BatchNormalizatio…	(None, 38, 38, 32)	128	block1a_conv1[
block1a_relu1 (Activation)	(None, 38, 38, 32)	0	block1a_bn1[0]
block1a_conv2 (Conv2D)	(None, 38, 38, 32)	9,248	block1a_relu1[
block1a_bn2 (BatchNormalizatio…	(None, 38, 38, 32)	128	block1a_conv2[
block1a_add (Add)	(None, 38, 38, 32)	0	block1a_bn2[0] pool1[0][0]
block1a_relu2 (Activation)	(None, 38, 38, 32)	0	block1a_add[0]
block1b_conv1 (Conv2D)	(None, 38, 38, 32)	9,248	block1a_relu2[
block1b_bn1 (BatchNormalizatio…	(None, 38, 38, 32)	128	block1b_conv1[
block1b_relu1 (Activation)	(None, 38, 38, 32)	0	block1b_bn1[0]
block1b_conv2 (Conv2D)	(None, 38, 38, 32)	9,248	block1b_relu1[
block1b_bn2 (BatchNormalizatio…	(None, 38, 38, 32)	128	block1b_conv2[
block1b_add (Add)	(None, 38, 38, 32)	0	block1b_bn2[0] block1a_relu2[
block1b_relu2 (Activation)	(None, 38, 38, 32)	0	block1b_add[0]
block2a_conv1 (Conv2D)	(None, 19, 19, 64)	18,496	block1b_relu2[
block2a_bn1 (BatchNormalizatio…	(None, 19, 19, 64)	256	block2a_conv1[

block2a_relu1 (Activation)	(None, 19, 19, 64)	0	block2a_bn1[0]
block2a_conv2 (Conv2D)	(None, 19, 19, 64)	36,928	block2a_relu1[
block2a_shortcut (Conv2D)	(None, 19, 19, 64)	2,112	block1b_relu2[
block2a_bn2 (BatchNormalizatio…	(None, 19, 19, 64)	256	block2a_conv2[
block2a_shortcut_bn (BatchNormalizatio…	(None, 19, 19, 64)	256	block2a_shortc
block2a_add (Add)	(None, 19, 19, 64)	0	block2a_bn2[0] block2a_shortc
block2a_relu2 (Activation)	(None, 19, 19, 64)	0	block2a_add[0]
block2b_conv1 (Conv2D)	(None, 19, 19, 64)	36,928	block2a_relu2[
block2b_bn1 (BatchNormalizatio…	(None, 19, 19, 64)	256	block2b_conv1[
block2b_relu1 (Activation)	(None, 19, 19, 64)	0	block2b_bn1[0]
block2b_conv2 (Conv2D)	(None, 19, 19, 64)	36,928	block2b_relu1[
block2b_bn2 (BatchNormalizatio…	(None, 19, 19, 64)	256	block2b_conv2[
block2b_add (Add)	(None, 19, 19, 64)	0	block2b_bn2[0] block2a_relu2[
block2b_relu2 (Activation)	(None, 19, 19, 64)	0	block2b_add[0]
block3a_conv1 (Conv2D)	(None, 10, 10, 128)	73,856	block2b_relu2[
block3a_bn1 (BatchNormalizatio…	(None, 10, 10, 128)	512	block3a_conv1[
block3a_relu1 (Activation)	(None, 10, 10, 128)	0	block3a_bn1[0]
block3a_conv2 (Conv2D)	(None, 10, 10, 128)	147,584	block3a_relu1[
block3a_shortcut (Conv2D)	(None, 10, 10, 128)	8,320	block2b_relu2[
block3a_bn2 (BatchNormalizatio…	(None, 10, 10, 128)	512	block3a_conv2[
block3a_shortcut_bn (BatchNormalizatio…	(None, 10, 10, 128)	512	block3a_shortc
block3a_add (Add)	(None, 10, 10,	0	block3a_bn2[0]

	128)		block3a_shortc
block3a_relu2 (Activation)	(None, 10, 10, 128)	0	block3a_add[0]
block3b_conv1 (Conv2D)	(None, 10, 10, 128)	147,584	block3a_relu2[
block3b_bn1 (BatchNormalizatio…	(None, 10, 10, 128)	512	block3b_conv1[
block3b_relu1 (Activation)	(None, 10, 10, 128)	0	block3b_bn1[0]
block3b_conv2 (Conv2D)	(None, 10, 10, 128)	147,584	block3b_relu1[
block3b_bn2 (BatchNormalizatio…	(None, 10, 10, 128)	512	block3b_conv2[
block3b_add (Add)	(None, 10, 10, 128)	0	block3b_bn2[0] block3a_relu2[
block3b_relu2 (Activation)	(None, 10, 10, 128)	0	block3b_add[0]
global_pool (GlobalAveragePool…	(None, 128)	0	block3b_relu2[
dropout (Dropout)	(None, 128)	0	global_pool[0]
output (Dense)	(None, 1)	129	dropout[0][0]

Total params: 702,657 (2.68 MB)

Trainable params: 700,417 (2.67 MB)

Non-trainable params: 2,240 (8.75 KB)

```
============================================================
TRAINING DATA INFO
============================================================
Training on BALANCED dataset:
  X_train_balanced: (2158, 150, 150, 3)
  y_train_balanced: (2158,)
  - Normal: 1079 (50%)
  - Pneumonia: 1079 (50%)

Validating on BALANCED dataset:
  X_val_balanced: (540, 150, 150, 3)
  y_val_balanced: (540,)
  - Normal: 270 (50%)
  - Pneumonia: 270 (50%)

============================================================
Training Model 2 on BALANCED dataset...
============================================================
Epoch 1/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 30s 174ms/step - accuracy: 0.8006 - loss: 0.4745
- val_accuracy: 0.5000 - val_loss: 1.6641 - learning_rate: 3.0000e-04
Epoch 2/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9449 - loss: 0.1499 -
val_accuracy: 0.5000 - val_loss: 4.7217 - learning_rate: 3.0000e-04
Epoch 3/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9589 - loss: 0.1171 -
val_accuracy: 0.5000 - val_loss: 5.3909 - learning_rate: 3.0000e-04
Epoch 4/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9785 - loss: 0.0626 -
val_accuracy: 0.5000 - val_loss: 3.9936 - learning_rate: 3.0000e-04
Epoch 5/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9848 - loss: 0.0372 -
val_accuracy: 0.5111 - val_loss: 2.0316 - learning_rate: 1.5000e-04
Epoch 6/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9935 - loss: 0.0207 -
val_accuracy: 0.6630 - val_loss: 0.9614 - learning_rate: 1.5000e-04
Epoch 7/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9968 - loss: 0.0185 -
val_accuracy: 0.8574 - val_loss: 0.3970 - learning_rate: 1.5000e-04
Epoch 8/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9933 - loss: 0.0238 -
val_accuracy: 0.9370 - val_loss: 0.1604 - learning_rate: 1.5000e-04
Epoch 9/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 1.0000 - loss: 0.0073 -
val_accuracy: 0.9722 - val_loss: 0.0670 - learning_rate: 1.5000e-04
Epoch 10/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 25ms/step - accuracy: 0.9910 - loss: 0.0257 -
val_accuracy: 0.9667 - val_loss: 0.0989 - learning_rate: 1.5000e-04
Epoch 11/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9944 - loss: 0.0204 -
val_accuracy: 0.9370 - val_loss: 0.1738 - learning_rate: 1.5000e-04
Epoch 12/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 26ms/step - accuracy: 0.9996 - loss: 0.0085 -
val_accuracy: 0.8944 - val_loss: 0.3107 - learning_rate: 1.5000e-04
Epoch 13/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 28ms/step - accuracy: 0.9995 - loss: 0.0074 -
val_accuracy: 0.9741 - val_loss: 0.0724 - learning_rate: 7.5000e-05
Epoch 14/15
68/68 ━━━━━━━━━━━━━━━━━━━━ 2s 27ms/step - accuracy: 0.9996 - loss: 0.0052 -
val_accuracy: 0.9741 - val_loss: 0.0705 - learning_rate: 7.5000e-05

Model 2 training completed in 55.87 seconds

============================================================
```

```
Evaluating Model 2 on Test Set...
============================================================
Testing on ORIGINAL (imbalanced) test set:
  X_test_scaled: (624, 150, 150, 3)
  - Normal: 234 (37.5%)
  - Pneumonia: 390 (62.5%)

Model 2 Results on Test Set:
  Overall Accuracy: 0.8365

  Class 0 (NORMAL) Metrics:
    Precision: 0.9648
    Recall:    0.5855
    F1 Score:  0.7287

  Class 1 (PNEUMONIA) Metrics:
    Precision: 0.7988
    Recall:    0.9872
    F1 Score:  0.8830

  Weighted Average:
    Precision: 0.8610
    Recall:    0.8365
    F1 Score:  0.8252

============================================================
```

In [7]:

```python
# ============================================
# MODEL 3: LIGHTWEIGHT VGG-STYLE CNN
# ============================================

print("\n" + "="*60)
print("MODEL 3: VGG")
print("="*60)

def create_lightweight_vgg(input_shape=(150, 150, 3)):
    """
    Lightweight VGG-style architecture for fast training
    - Fewer filters than standard VGG
    - Fewer blocks
    - Efficient design
    """
    model = Sequential(name='lightweight_vgg', layers=[
        # Block 1: 32 filters
        Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=in
        Conv2D(32, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        # Block 2: 64 filters
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        Conv2D(64, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        # Block 3: 128 filters
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),

        # Block 4: 256 filters
        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D((2, 2)),
```

```
        # Classifier
        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(128, activation='relu'),
        Dropout(0.4),
        Dense(1, activation='sigmoid')
    ])

    return model

# Create and compile Model 3
model3 = create_lightweight_vgg()
model3.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

print("\nModel 3 Architecture:")
model3.summary()

# Display training data info
print("\n" + "="*60)
print("TRAINING DATA INFO")
print("="*60)
print(f"Training on BALANCED dataset:")
print(f"  X_train_balanced: {X_train_balanced.shape}")
print(f"  y_train_balanced: {y_train_balanced.shape}")
print(f"  - Normal: {np.sum(y_train_balanced == 0)} (50%)")
print(f"  - Pneumonia: {np.sum(y_train_balanced == 1)} (50%)")
print(f"\nValidating on BALANCED dataset:")
print(f"  X_val_balanced: {X_val_balanced.shape}")
print(f"  y_val_balanced: {y_val_balanced.shape}")
print(f"  - Normal: {np.sum(y_val_balanced == 0)} (50%)")
print(f"  - Pneumonia: {np.sum(y_val_balanced == 1)} (50%)")

# Train Model 3
print("\nTraining Model 3...")
start_time = time.time()

history3 = model3.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=15,
    batch_size=32,
    callbacks=[
        EarlyStopping(monitor='val_loss', patience=4, restore_best_weights=T
        ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr
    ],
    verbose=1
)

model3_time = time.time() - start_time
print(f"\nModel 3 training completed in {model3_time:.2f} seconds")

# Evaluate Model 3
print("\nEvaluating Model 3 on Test Set...")
y_pred3_prob = model3.predict(X_test_scaled, verbose=0)
y_pred3 = (y_pred3_prob > 0.5).astype(int).flatten()

# Calculate metrics for both classes
f1_model3 = f1_score(y_test, y_pred3, average=None)  # Per-class F1
f1_weighted_model3 = f1_score(y_test, y_pred3, average='weighted')  # Weight
```

```
accuracy_model3 = accuracy_score(y_test, y_pred3)  # Overall accuracy

precision_model3 = precision_score(y_test, y_pred3, average=None)  # Per-cla
precision_weighted_model3 = precision_score(y_test, y_pred3, average='weight

recall_model3 = recall_score(y_test, y_pred3, average=None)  # Per-class rec
recall_weighted_model3 = recall_score(y_test, y_pred3, average='weighted')

print(f"\nModel 3 Results on Test Set:")
print(f"  Overall Accuracy: {accuracy_model3:.4f}")
print(f"\n  Class 0 (NORMAL) Metrics:")
print(f"    Precision: {precision_model3[0]:.4f}")
print(f"    Recall:    {recall_model3[0]:.4f}")
print(f"    F1 Score:  {f1_model3[0]:.4f}")
print(f"\n  Class 1 (PNEUMONIA) Metrics:")
print(f"    Precision: {precision_model3[1]:.4f}")
print(f"    Recall:    {recall_model3[1]:.4f}")
print(f"    F1 Score:  {f1_model3[1]:.4f}")
print(f"\n  Weighted Average:")
print(f"    Precision: {precision_weighted_model3:.4f}")
print(f"    Recall:    {recall_weighted_model3:.4f}")
print(f"    F1 Score:  {f1_weighted_model3:.4f}")

print("\n" + "="*60)
print("MODEL 3 COMPLETED!")
print("="*60)
```

```
============================================================
MODEL 3: VGG
============================================================

Model 3 Architecture:
Model: "lightweight_vgg"
```

Layer (type)	Output Shape	Para
conv2d_3 (Conv2D)	(None, 150, 150, 32)	
conv2d_4 (Conv2D)	(None, 150, 150, 32)	9,
max_pooling2d_3 (MaxPooling2D)	(None, 75, 75, 32)	
conv2d_5 (Conv2D)	(None, 75, 75, 64)	18,
conv2d_6 (Conv2D)	(None, 75, 75, 64)	36,
max_pooling2d_4 (MaxPooling2D)	(None, 37, 37, 64)	
conv2d_7 (Conv2D)	(None, 37, 37, 128)	73,
conv2d_8 (Conv2D)	(None, 37, 37, 128)	147,
max_pooling2d_5 (MaxPooling2D)	(None, 18, 18, 128)	
conv2d_9 (Conv2D)	(None, 18, 18, 256)	295,
max_pooling2d_6 (MaxPooling2D)	(None, 9, 9, 256)	
flatten_1 (Flatten)	(None, 20736)	
dense_2 (Dense)	(None, 256)	5,308,
dropout_1 (Dropout)	(None, 256)	
dense_3 (Dense)	(None, 128)	32,
dropout_2 (Dropout)	(None, 128)	
dense_4 (Dense)	(None, 1)	

Total params: 5,923,873 (22.60 MB)
Trainable params: 5,923,873 (22.60 MB)
Non-trainable params: 0 (0.00 B)

```
============================================================
TRAINING DATA INFO
============================================================
Training on BALANCED dataset:
  X_train_balanced: (2158, 150, 150, 3)
  y_train_balanced: (2158,)
  - Normal: 1079 (50%)
  - Pneumonia: 1079 (50%)

Validating on BALANCED dataset:
  X_val_balanced: (540, 150, 150, 3)
  y_val_balanced: (540,)
  - Normal: 270 (50%)
  - Pneumonia: 270 (50%)

Training Model 3...
Epoch 1/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 40s 186ms/step - accuracy: 0.7237 - loss: 0.66
29 - val_accuracy: 0.7421 - val_loss: 0.5450 - learning_rate: 0.0010
Epoch 2/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 69ms/step - accuracy: 0.7893 - loss: 0.4486
- val_accuracy: 0.9331 - val_loss: 0.1909 - learning_rate: 0.0010
Epoch 3/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 70ms/step - accuracy: 0.9300 - loss: 0.1913
- val_accuracy: 0.9532 - val_loss: 0.1108 - learning_rate: 0.0010
Epoch 4/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 10s 70ms/step - accuracy: 0.9566 - loss: 0.111
0 - val_accuracy: 0.9580 - val_loss: 0.1042 - learning_rate: 0.0010
Epoch 5/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 71ms/step - accuracy: 0.9673 - loss: 0.0889
- val_accuracy: 0.9608 - val_loss: 0.1039 - learning_rate: 0.0010
Epoch 6/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 71ms/step - accuracy: 0.9724 - loss: 0.0697
- val_accuracy: 0.9608 - val_loss: 0.1220 - learning_rate: 0.0010
Epoch 7/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 71ms/step - accuracy: 0.9737 - loss: 0.0778
- val_accuracy: 0.9647 - val_loss: 0.1188 - learning_rate: 0.0010
Epoch 8/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 70ms/step - accuracy: 0.9888 - loss: 0.0344
- val_accuracy: 0.9761 - val_loss: 0.0658 - learning_rate: 5.0000e-04
Epoch 9/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 70ms/step - accuracy: 0.9953 - loss: 0.0156
- val_accuracy: 0.9589 - val_loss: 0.1804 - learning_rate: 5.0000e-04
Epoch 10/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 10s 73ms/step - accuracy: 0.9882 - loss: 0.031
2 - val_accuracy: 0.9752 - val_loss: 0.0904 - learning_rate: 5.0000e-04
Epoch 11/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 70ms/step - accuracy: 0.9973 - loss: 0.0066
- val_accuracy: 0.9742 - val_loss: 0.1078 - learning_rate: 2.5000e-04
Epoch 12/15
131/131 ━━━━━━━━━━━━━━━━━━━━ 9s 70ms/step - accuracy: 0.9994 - loss: 0.0042
- val_accuracy: 0.9780 - val_loss: 0.1067 - learning_rate: 2.5000e-04

Model 3 training completed in 146.10 seconds

Evaluating Model 3 on Test Set...

Model 3 Results on Test Set:
  Overall Accuracy: 0.7612

  Class 0 (NORMAL) Metrics:
    Precision: 0.9885
    Recall:    0.3675
    F1 Score:  0.5358
```

```
  Class 1 (PNEUMONIA) Metrics:
    Precision: 0.7244
    Recall:    0.9974
    F1 Score:  0.8393

  Weighted Average:
    Precision: 0.8234
    Recall:    0.7612
    F1 Score:  0.7255

============================================================
```

In [24]:

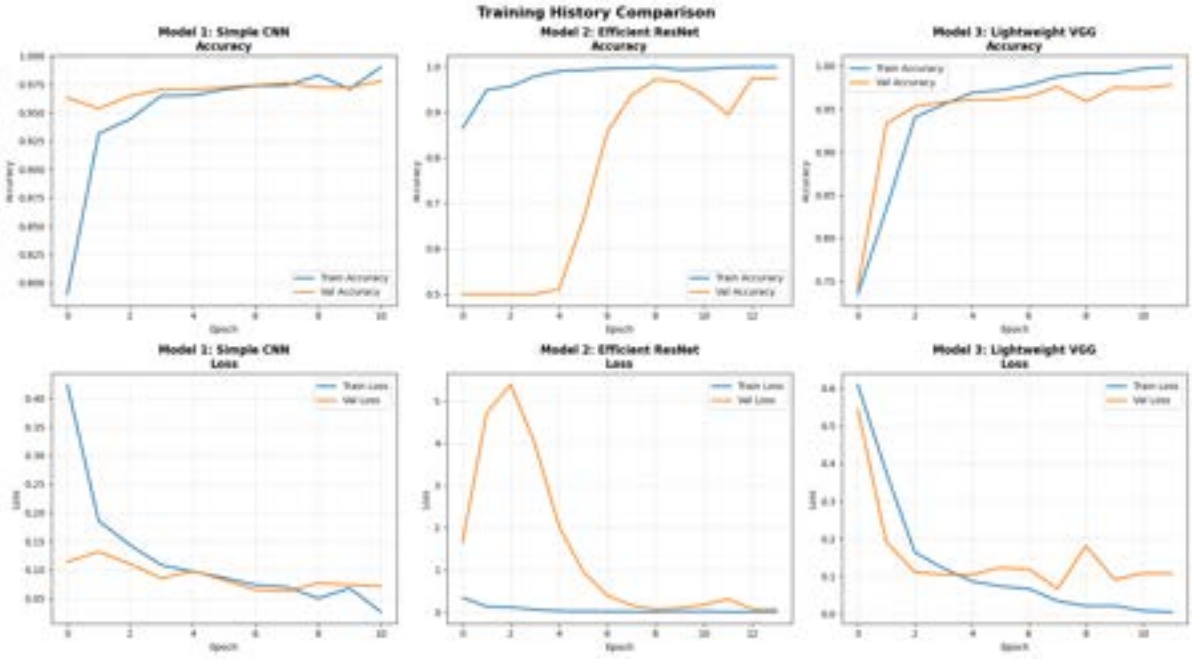
```
# ============================================
# MODEL COMPARISON
# ============================================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report

# ============================================
# STEP 1: CREATE COMPARISON DATAFRAME
# ============================================

print("\n" + "="*60)
print("CREATING MODEL COMPARISON DATAFRAME")
print("="*60)

# Create comparison dataframe with WEIGHTED AVERAGE metrics
comparison_data = {
    'Model': [
        'Model 1: Simple CNN',
        'Model 2: ResNet',
        'Model 3: VGG'
    ],
    'Accuracy': [accuracy_model1, accuracy_model2, accuracy_model3],
    'Precision (Weighted)': [precision_weighted_model1, precision_weighted_m
    'Recall (Weighted)': [recall_weighted_model1, recall_weighted_model2, re
    'F1 Score (Weighted)': [f1_weighted_model1, f1_weighted_model2, f1_weigh
    'Training Time (s)': [model1_time, model2_time, model3_time]
}

comparison_df = pd.DataFrame(comparison_data)

# Display the comparison table
print("\n📊 Model Performance Comparison (Weighted Averages):")
print(comparison_df.to_string(index=False))

# Create detailed per-class comparison
print("\n📊 Detailed Per-Class Comparison:")
print("\n" + "="*60)

for model_num, model_name in enumerate(['Model 1: Simple CNN', 'Model 2: Res
    print(f"\n{model_name}:")
    if model_num == 1:
        print(f"  Class 0 (NORMAL):    Precision={precision_model1[0]:.4f},
        print(f"  Class 1 (PNEUMONIA): Precision={precision_model1[1]:.4f},
    elif model_num == 2:
        print(f"  Class 0 (NORMAL):    Precision={precision_model2[0]:.4f},
```

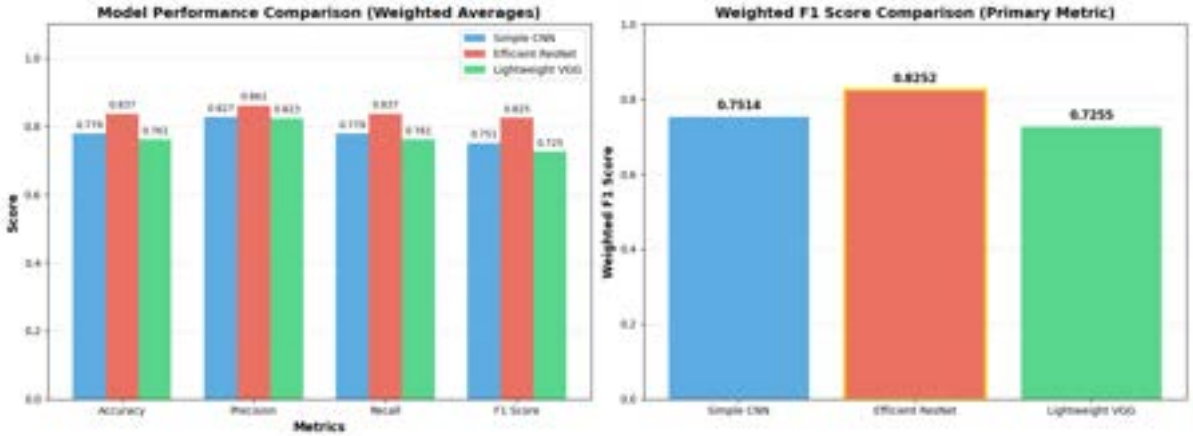
```
        print(f"  Class 1 (PNEUMONIA): Precision={precision_model2[1]:.4f},
    else:
        print(f"  Class 0 (NORMAL):    Precision={precision_model3[0]:.4f},
        print(f"  Class 1 (PNEUMONIA): Precision={precision_model3[1]:.4f},

# Find best model based on Weighted F1 Score
best_model_idx = comparison_df['F1 Score (Weighted)'].idxmax()
best_model_name = comparison_df.loc[best_model_idx, 'Model']
best_f1 = comparison_df.loc[best_model_idx, 'F1 Score (Weighted)']

print("\n" + "="*60)
print(f"🏆 Best Model: {best_model_name} (Weighted F1: {best_f1:.4f})")
print("="*60)

# ============================================
# STEP 2: VISUALIZATIONS
# ============================================

# 1. Training History Comparison
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
fig.suptitle('Training History Comparison', fontsize=16, fontweight='bold')

models_history = [
    (history1, 'Model 1: Simple CNN'),
    (history2, 'Model 2: Efficient ResNet'),
    (history3, 'Model 3: Lightweight VGG')
]

for idx, (history, title) in enumerate(models_history):
    # Training & Validation Accuracy
    axes[0, idx].plot(history.history['accuracy'], label='Train Accuracy', 
    axes[0, idx].plot(history.history['val_accuracy'], label='Val Accuracy',
    axes[0, idx].set_title(f'{title}\nAccuracy', fontweight='bold')
    axes[0, idx].set_xlabel('Epoch')
    axes[0, idx].set_ylabel('Accuracy')
    axes[0, idx].legend()
    axes[0, idx].grid(alpha=0.3)

    # Training & Validation Loss
    axes[1, idx].plot(history.history['loss'], label='Train Loss', linewidth
    axes[1, idx].plot(history.history['val_loss'], label='Val Loss', linewid
    axes[1, idx].set_title(f'{title}\nLoss', fontweight='bold')
    axes[1, idx].set_xlabel('Epoch')
    axes[1, idx].set_ylabel('Loss')
    axes[1, idx].legend()
    axes[1, idx].grid(alpha=0.3)

plt.tight_layout()
plt.show()

# 2. Model Performance Comparison Bar Chart (Weighted Averages)
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
x = np.arange(len(metrics))
width = 0.25

model1_scores = [accuracy_model1, precision_weighted_model1, recall_weighted
model2_scores = [accuracy_model2, precision_weighted_model2, recall_weighted
model3_scores = [accuracy_model3, precision_weighted_model3, recall_weighted

bars1 = axes[0].bar(x - width, model1_scores, width, label='Simple CNN', alp
bars2 = axes[0].bar(x, model2_scores, width, label='Efficient ResNet', alpha
bars3 = axes[0].bar(x + width, model3_scores, width, label='Lightweight VGG
```

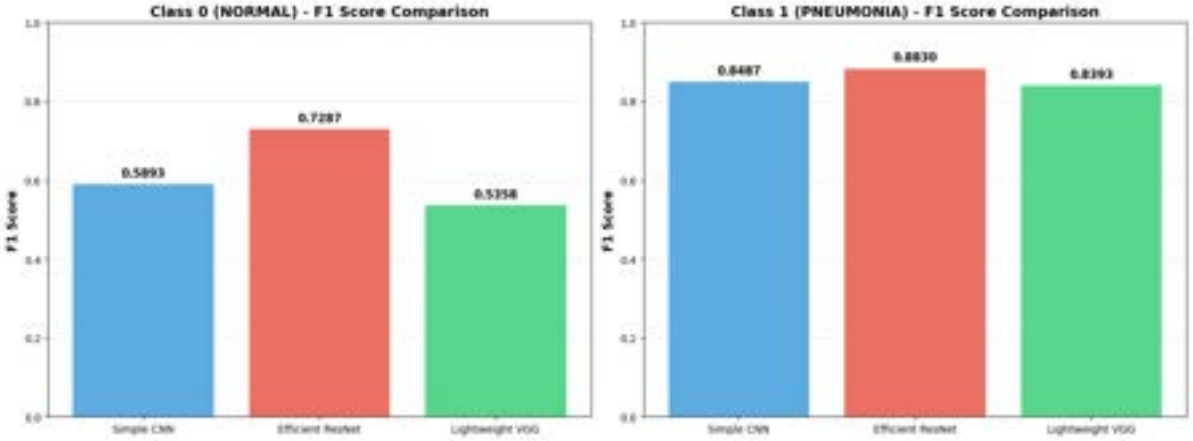
```
# Add value labels on bars
for bars in [bars1, bars2, bars3]:
    for bar in bars:
        height = bar.get_height()
        axes[0].text(bar.get_x() + bar.get_width()/2., height + 0.01,
                     f'{height:.3f}', ha='center', va='bottom', fontsize=9)

axes[0].set_xlabel('Metrics', fontweight='bold', fontsize=12)
axes[0].set_ylabel('Score', fontweight='bold', fontsize=12)
axes[0].set_title('Model Performance Comparison (Weighted Averages)', fontwe
axes[0].set_xticks(x)
axes[0].set_xticklabels(metrics)
axes[0].legend()
axes[0].grid(axis='y', alpha=0.3)
axes[0].set_ylim([0, 1.1])

# Weighted F1 Score Comparison
f1_scores = [f1_weighted_model1, f1_weighted_model2, f1_weighted_model3]
colors = ['#3498db', '#e74c3c', '#2ecc71']
bars = axes[1].bar(['Simple CNN', 'Efficient ResNet', 'Lightweight VGG'],
                   f1_scores, color=colors, alpha=0.8)
axes[1].set_ylabel('Weighted F1 Score', fontweight='bold', fontsize=12)
axes[1].set_title('Weighted F1 Score Comparison (Primary Metric)', fontweigh
axes[1].grid(axis='y', alpha=0.3)
axes[1].set_ylim([0, 1.0])

# Add value labels and highlight best
for i, v in enumerate(f1_scores):
    axes[1].text(i, v + 0.02, f'{v:.4f}', ha='center', fontweight='bold', fo
    if i == best_model_idx:
        bars[i].set_linewidth(3)
        bars[i].set_edgecolor('gold')

plt.tight_layout()
plt.show()

# 3. Per-Class F1 Score Comparison
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Class 0 (NORMAL) F1 Scores
class0_f1 = [f1_model1[0], f1_model2[0], f1_model3[0]]
bars = axes[0].bar(['Simple CNN', 'Efficient ResNet', 'Lightweight VGG'],
                   class0_f1, color=colors, alpha=0.8)
axes[0].set_ylabel('F1 Score', fontweight='bold', fontsize=12)
axes[0].set_title('Class 0 (NORMAL) - F1 Score Comparison', fontweight='bold
axes[0].grid(axis='y', alpha=0.3)
axes[0].set_ylim([0, 1.0])

for i, v in enumerate(class0_f1):
    axes[0].text(i, v + 0.02, f'{v:.4f}', ha='center', fontweight='bold', fo

# Class 1 (PNEUMONIA) F1 Scores
class1_f1 = [f1_model1[1], f1_model2[1], f1_model3[1]]
bars = axes[1].bar(['Simple CNN', 'Efficient ResNet', 'Lightweight VGG'],
                   class1_f1, color=colors, alpha=0.8)
axes[1].set_ylabel('F1 Score', fontweight='bold', fontsize=12)
axes[1].set_title('Class 1 (PNEUMONIA) - F1 Score Comparison', fontweight='b
axes[1].grid(axis='y', alpha=0.3)
axes[1].set_ylim([0, 1.0])

for i, v in enumerate(class1_f1):
    axes[1].text(i, v + 0.02, f'{v:.4f}', ha='center', fontweight='bold', fo
```

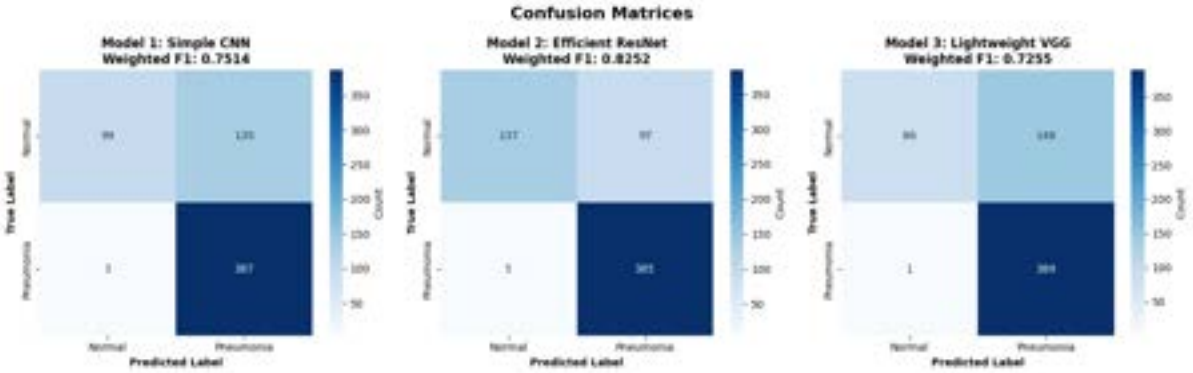
```
plt.tight_layout()
plt.show()

# 4. Confusion Matrices
fig, axes = plt.subplots(1, 3, figsize=(16, 5))
fig.suptitle('Confusion Matrices', fontsize=16, fontweight='bold')

predictions = [
    (y_pred1, 'Model 1: Simple CNN', f1_weighted_model1),
    (y_pred2, 'Model 2: Efficient ResNet', f1_weighted_model2),
    (y_pred3, 'Model 3: Lightweight VGG', f1_weighted_model3)
]

for idx, (y_pred, title, f1) in enumerate(predictions):
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[idx],
                xticklabels=['Normal', 'Pneumonia'],
                yticklabels=['Normal', 'Pneumonia'],
                cbar_kws={'label': 'Count'})
    axes[idx].set_title(f'{title}\nWeighted F1: {f1:.4f}', fontweight='bold
    axes[idx].set_ylabel('True Label', fontweight='bold')
    axes[idx].set_xlabel('Predicted Label', fontweight='bold')

plt.tight_layout()
plt.show()
```

```
============================================================
CREATING MODEL COMPARISON DATAFRAME
============================================================

📊 Model Performance Comparison (Weighted Averages):
              Model  Accuracy  Precision (Weighted)  Recall (Weighted)  F1
Score (Weighted)  Training Time (s)
Model 1: Simple CNN  0.778846              0.827333           0.778846
0.751410          32.284436
    Model 2: ResNet  0.836538              0.861018           0.836538
0.825163          55.874117
       Model 3: VGG  0.761218              0.823436           0.761218
0.725476         146.101899

📊 Detailed Per-Class Comparison:

============================================================

Model 1: Simple CNN:
  Class 0 (NORMAL):    Precision=0.9706, Recall=0.4231, F1=0.5893
  Class 1 (PNEUMONIA): Precision=0.7414, Recall=0.9923, F1=0.8487

Model 2: ResNet:
  Class 0 (NORMAL):    Precision=0.9648, Recall=0.5855, F1=0.7287
  Class 1 (PNEUMONIA): Precision=0.7988, Recall=0.9872, F1=0.8830

Model 3: VGG:
  Class 0 (NORMAL):    Precision=0.9885, Recall=0.3675, F1=0.5358
  Class 1 (PNEUMONIA): Precision=0.7244, Recall=0.9974, F1=0.8393

============================================================
🏆 Best Model: Model 2: ResNet (Weighted F1: 0.8252)
============================================================
```









In [ ]:
```
# ============================================
# MODEL COMPARISON WITH ROC AUC
# ============================================
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, roc_cu

# ============================================
# STEP 1: CALCULATE ROC AUC FOR ALL MODELS
# ============================================

print("\n" + "="*60)
print("CALCULATING ROC AUC SCORES")
print("="*60)

# Calculate ROC AUC for each model
roc_auc_model1 = roc_auc_score(y_test, y_pred1_prob)
roc_auc_model2 = roc_auc_score(y_test, y_pred2_prob)
roc_auc_model3 = roc_auc_score(y_test, y_pred3_prob)

print(f"Model 1 (Simple CNN):     ROC AUC = {roc_auc_model1:.4f}")
print(f"Model 2 (ResNet):         ROC AUC = {roc_auc_model2:.4f}")
print(f"Model 3 (VGG):            ROC AUC = {roc_auc_model3:.4f}")

# Calculate ROC curves
fpr_model1, tpr_model1, thresholds_model1 = roc_curve(y_test, y_pred1_prob)
fpr_model2, tpr_model2, thresholds_model2 = roc_curve(y_test, y_pred2_prob)
fpr_model3, tpr_model3, thresholds_model3 = roc_curve(y_test, y_pred3_prob)

# ============================================
# STEP 2: CREATE COMPARISON DATAFRAME
# ============================================

print("\n" + "="*60)
print("CREATING MODEL COMPARISON DATAFRAME")
print("="*60)

# Create comparison dataframe with WEIGHTED AVERAGE metrics + ROC AUC
comparison_data = {
    'Model': [
        'Model 1: Simple CNN',
        'Model 2: ResNet',
        'Model 3: VGG'
    ],
    'Accuracy': [accuracy_model1, accuracy_model2, accuracy_model3],
    'Precision (Weighted)': [precision_weighted_model1, precision_weighted_m
    'Recall (Weighted)': [recall_weighted_model1, recall_weighted_model2, re
    'F1 Score (Weighted)': [f1_weighted_model1, f1_weighted_model2, f1_weigh
    'ROC AUC': [roc_auc_model1, roc_auc_model2, roc_auc_model3],
    'Training Time (s)': [model1_time, model2_time, model3_time]
}

comparison_df = pd.DataFrame(comparison_data)

# Display the comparison table
print("\n📊 Model Performance Comparison (Weighted Averages + ROC AUC):")
print(comparison_df.to_string(index=False))

# Create detailed per-class comparison
print("\n📊 Detailed Per-Class Comparison:")
print("\n" + "="*60)

for model_num, model_name in enumerate(['Model 1: Simple CNN', 'Model 2: Res
    print(f"\n{model_name}:")
```

```
    if model_num == 1:
        print(f"  Class 0 (NORMAL):    Precision={precision_model1[0]:.4f},
        print(f"  Class 1 (PNEUMONIA): Precision={precision_model1[1]:.4f},
        print(f"  ROC AUC: {roc_auc_model1:.4f}")
    elif model_num == 2:
        print(f"  Class 0 (NORMAL):    Precision={precision_model2[0]:.4f},
        print(f"  Class 1 (PNEUMONIA): Precision={precision_model2[1]:.4f},
        print(f"  ROC AUC: {roc_auc_model2:.4f}")
    else:
        print(f"  Class 0 (NORMAL):    Precision={precision_model3[0]:.4f},
        print(f"  Class 1 (PNEUMONIA): Precision={precision_model3[1]:.4f},
        print(f"  ROC AUC: {roc_auc_model3:.4f}")

# Find best model based on Weighted F1 Score
best_model_idx = comparison_df['F1 Score (Weighted)'].idxmax()
best_model_name = comparison_df.loc[best_model_idx, 'Model']
best_f1 = comparison_df.loc[best_model_idx, 'F1 Score (Weighted)']

# Find best model based on ROC AUC
best_roc_idx = comparison_df['ROC AUC'].idxmax()
best_roc_name = comparison_df.loc[best_roc_idx, 'Model']
best_roc = comparison_df.loc[best_roc_idx, 'ROC AUC']

print("\n" + "="*60)
print(f"🏆 Best Model (Weighted F1): {best_model_name} (F1: {best_f1:.4f})"
print(f"🏆 Best Model (ROC AUC):     {best_roc_name} (AUC: {best_roc:.4f})"
print("="*60)

# ============================================
# STEP 3: VISUALIZATIONS
# ============================================

# 1. ROC Curves Comparison
plt.figure(figsize=(10, 8))

plt.plot(fpr_model1, tpr_model1, linewidth=2, label=f'Model 1: Simple CNN (A
plt.plot(fpr_model2, tpr_model2, linewidth=2, label=f'Model 2: ResNet (AUC =
plt.plot(fpr_model3, tpr_model3, linewidth=2, label=f'Model 3: VGG (AUC = {r
plt.plot([0, 1], [0, 1], 'k--', linewidth=1, label='Random Classifier (AUC =

plt.xlabel('False Positive Rate', fontsize=12, fontweight='bold')
plt.ylabel('True Positive Rate', fontsize=12, fontweight='bold')
plt.title('ROC Curves Comparison', fontsize=14, fontweight='bold')
plt.legend(loc='lower right', fontsize=11)
plt.grid(alpha=0.3)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.tight_layout()
plt.show()

# 2. Training History Comparison
fig, axes = plt.subplots(2, 3, figsize=(18, 10))
fig.suptitle('Training History Comparison', fontsize=16, fontweight='bold')

models_history = [
    (history1, 'Model 1: Simple CNN'),
    (history2, 'Model 2: Efficient ResNet'),
    (history3, 'Model 3: Lightweight VGG')
]

for idx, (history, title) in enumerate(models_history):
    # Training & Validation Accuracy
    axes[0, idx].plot(history.history['accuracy'], label='Train Accuracy', l
    axes[0, idx].plot(history.history['val_accuracy'], label='Val Accuracy',
```

```
    axes[0, idx].set_title(f'{title}\nAccuracy', fontweight='bold')
    axes[0, idx].set_xlabel('Epoch')
    axes[0, idx].set_ylabel('Accuracy')
    axes[0, idx].legend()
    axes[0, idx].grid(alpha=0.3)

    # Training & Validation Loss
    axes[1, idx].plot(history.history['loss'], label='Train Loss', linewidth
    axes[1, idx].plot(history.history['val_loss'], label='Val Loss', linewid
    axes[1, idx].set_title(f'{title}\nLoss', fontweight='bold')
    axes[1, idx].set_xlabel('Epoch')
    axes[1, idx].set_ylabel('Loss')
    axes[1, idx].legend()
    axes[1, idx].grid(alpha=0.3)

plt.tight_layout()
plt.show()

# 3. Model Performance Comparison Bar Chart (Weighted Averages + ROC AUC)
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC']
x = np.arange(len(metrics))
width = 0.25

model1_scores = [accuracy_model1, precision_weighted_model1, recall_weighted
model2_scores = [accuracy_model2, precision_weighted_model2, recall_weighted
model3_scores = [accuracy_model3, precision_weighted_model3, recall_weighted

bars1 = axes[0].bar(x - width, model1_scores, width, label='Simple CNN', alp
bars2 = axes[0].bar(x, model2_scores, width, label='Efficient ResNet', alpha
bars3 = axes[0].bar(x + width, model3_scores, width, label='Lightweight VGG

# Add value labels on bars
for bars in [bars1, bars2, bars3]:
    for bar in bars:
        height = bar.get_height()
        axes[0].text(bar.get_x() + bar.get_width()/2., height + 0.01,
                     f'{height:.3f}', ha='center', va='bottom', fontsize=9)

axes[0].set_xlabel('Metrics', fontweight='bold', fontsize=12)
axes[0].set_ylabel('Score', fontweight='bold', fontsize=12)
axes[0].set_title('Model Performance Comparison (Including ROC AUC)', fontwe
axes[0].set_xticks(x)
axes[0].set_xticklabels(metrics, rotation=15)
axes[0].legend()
axes[0].grid(axis='y', alpha=0.3)
axes[0].set_ylim([0, 1.1])

# ROC AUC Comparison
roc_scores = [roc_auc_model1, roc_auc_model2, roc_auc_model3]
colors = ['#3498db', '#e74c3c', '#2ecc71']
bars = axes[1].bar(['Simple CNN', 'Efficient ResNet', 'Lightweight VGG'],
                   roc_scores, color=colors, alpha=0.8)
axes[1].set_ylabel('ROC AUC Score', fontweight='bold', fontsize=12)
axes[1].set_title('ROC AUC Score Comparison', fontweight='bold', fontsize=14
axes[1].grid(axis='y', alpha=0.3)
axes[1].set_ylim([0, 1.0])

# Add value labels and highlight best
for i, v in enumerate(roc_scores):
    axes[1].text(i, v + 0.02, f'{v:.4f}', ha='center', fontweight='bold', fo
    if i == best_roc_idx:
        bars[i].set_linewidth(3)
```

```
        bars[i].set_edgecolor('gold')

plt.tight_layout()
plt.show()

# 4. Per-Class F1 Score Comparison
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Class 0 (NORMAL) F1 Scores
class0_f1 = [f1_model1[0], f1_model2[0], f1_model3[0]]
bars = axes[0].bar(['Simple CNN', 'Efficient ResNet', 'Lightweight VGG'],
                   class0_f1, color=colors, alpha=0.8)
axes[0].set_ylabel('F1 Score', fontweight='bold', fontsize=12)
axes[0].set_title('Class 0 (NORMAL) - F1 Score Comparison', fontweight='bol
axes[0].grid(axis='y', alpha=0.3)
axes[0].set_ylim([0, 1.0])

for i, v in enumerate(class0_f1):
    axes[0].text(i, v + 0.02, f'{v:.4f}', ha='center', fontweight='bold', fo

# Class 1 (PNEUMONIA) F1 Scores
class1_f1 = [f1_model1[1], f1_model2[1], f1_model3[1]]
bars = axes[1].bar(['Simple CNN', 'Efficient ResNet', 'Lightweight VGG'],
                   class1_f1, color=colors, alpha=0.8)
axes[1].set_ylabel('F1 Score', fontweight='bold', fontsize=12)
axes[1].set_title('Class 1 (PNEUMONIA) - F1 Score Comparison', fontweight='b
axes[1].grid(axis='y', alpha=0.3)
axes[1].set_ylim([0, 1.0])

for i, v in enumerate(class1_f1):
    axes[1].text(i, v + 0.02, f'{v:.4f}', ha='center', fontweight='bold', fo

plt.tight_layout()
plt.show()

# 5. Confusion Matrices
fig, axes = plt.subplots(1, 3, figsize=(16, 5))
fig.suptitle('Confusion Matrices', fontsize=16, fontweight='bold')

predictions = [
    (y_pred1, 'Model 1: Simple CNN', f1_weighted_model1, roc_auc_model1),
    (y_pred2, 'Model 2: Efficient ResNet', f1_weighted_model2, roc_auc_model
    (y_pred3, 'Model 3: Lightweight VGG', f1_weighted_model3, roc_auc_model3
]

for idx, (y_pred, title, f1, roc_auc) in enumerate(predictions):
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[idx],
                xticklabels=['Normal', 'Pneumonia'],
                yticklabels=['Normal', 'Pneumonia'],
                cbar_kws={'label': 'Count'})
    axes[idx].set_title(f'{title}\nF1: {f1:.4f} | AUC: {roc_auc:.4f}', fontw
    axes[idx].set_ylabel('True Label', fontweight='bold')
    axes[idx].set_xlabel('Predicted Label', fontweight='bold')

plt.tight_layout()
plt.show()

print("\n" + "="*60)
print("MODEL COMPARISON COMPLETE!")
print("="*60)
```

In [9]:

```
# ============================================
# SAVE TRAINED MODELS
```

```
# ============================================

print("\n" + "="*60)
print("SAVING MODELS...")
print("="*60)

# Save Model 1
model1.save('simple_cnn_model.h5')
print("✓ Model 1 saved as: simple_cnn_model.h5")

# Save Model 2
# model2.save('resnet_model.h5')
# print("✓ Model 2 saved as: resnet_model.h5")

# # Save Model 3
# model3.save('vgg_model.h5')
# print("✓ Model 3 saved as: vgg_model.h5")

# Save metrics for reference (with per-class and weighted averages)
import json

metrics = {
    'model1': {
        'accuracy': float(accuracy_model1),
        'class_0_normal': {
            'precision': float(precision_model1[0]),
            'recall': float(recall_model1[0]),
            'f1_score': float(f1_model1[0])
        },
        'class_1_pneumonia': {
            'precision': float(precision_model1[1]),
            'recall': float(recall_model1[1]),
            'f1_score': float(f1_model1[1])
        },
        'weighted_average': {
            'precision': float(precision_weighted_model1),
            'recall': float(recall_weighted_model1),
            'f1_score': float(f1_weighted_model1)
        },
        'training_time': float(model1_time)
    },
    # 'model2': {
    #     'accuracy': float(accuracy_model2),
    #     'class_0_normal': {
    #         'precision': float(precision_model2[0]),
    #         'recall': float(recall_model2[0]),
    #         'f1_score': float(f1_model2[0])
    #     },
    #     'class_1_pneumonia': {
    #         'precision': float(precision_model2[1]),
    #         'recall': float(recall_model2[1]),
    #         'f1_score': float(f1_model2[1])
    #     },
    #     'weighted_average': {
    #         'precision': float(precision_weighted_model2),
    #         'recall': float(recall_weighted_model2),
    #         'f1_score': float(f1_weighted_model2)
    #     },
    #     'training_time': float(model2_time)
    # },
    # 'model3': {
    #     'accuracy': float(accuracy_model3),
    #     'class_0_normal': {
    #         'precision': float(precision_model3[0]),
```

```
    #         'recall': float(recall_model3[0]),
    #         'f1_score': float(f1_model3[0])
    #     },
    #     'class_1_pneumonia': {
    #         'precision': float(precision_model3[1]),
    #         'recall': float(recall_model3[1]),
    #         'f1_score': float(f1_model3[1])
    #     },
    #     'weighted_average': {
    #         'precision': float(precision_weighted_model3),
    #         'recall': float(recall_weighted_model3),
    #         'f1_score': float(f1_weighted_model3)
    #     },
    #     'training_time': float(model3_time)
    # }
}

with open('model_metrics.json', 'w') as f:
    json.dump(metrics, f, indent=4)
print("✓ Metrics saved as: model_metrics.json")

print("\n" + "="*60)
print("ALL MODELS SAVED SUCCESSFULLY!")
print("="*60)

# Print summary of saved metrics
print("\n📊 Saved Metrics Summary:")
for model_name, model_metrics in metrics.items():
    print(f"\n{model_name.upper()}:")
    print(f"  Accuracy: {model_metrics['accuracy']:.4f}")
    print(f"  Weighted F1: {model_metrics['weighted_average']['f1_score']:.4
    print(f"  Training Time: {model_metrics['training_time']:.2f}s")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` o
r `keras.saving.save_model(model)`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g. `model.save('my_mo
del.keras')` or `keras.saving.save_model(model, 'my_model.keras')`.
============================================================
SAVING MODELS...
============================================================
✓ Model 1 saved as: simple_cnn_model.h5
✓ Metrics saved as: model_metrics.json

============================================================
ALL MODELS SAVED SUCCESSFULLY!
============================================================

📊 Saved Metrics Summary:

MODEL1:
  Accuracy: 0.7788
  Weighted F1: 0.7514
  Training Time: 32.28s
```

In [10]:

```
print("\n" + "="*60)
print("GRAD-CAM ANALYSIS - ALL 3 MODELS")
print("Enhanced heatmap for Model 3 (VGG16)")
print("="*60)

# ============================================
# STEP 1: GRAD-CAM CLASSES (Standard + Enhanced)
# ============================================

class GradCAM:
```

```
    """Standard Grad-CAM for Models 1 and 2"""
    def __init__(self, model, layer_name, model_name="Model"):
        self.model = model
        self.layer_name = layer_name
        self.model_name = model_name

        # Ensure the model is built
        if not model.built:
            print(f"⚠️  {model_name}: Building model...")
            model.build(input_shape=(None, 150, 150, 3))

        # Get the convolutional layer
        try:
            self.conv_layer = model.get_layer(layer_name)
        except ValueError as e:
            print(f"❌ {model_name}: Error getting layer '{layer_name}': {e
            # List available conv layers
            conv_layers = [layer.name for layer in model.layers if 'conv' in
            print(f"   Available conv layers: {conv_layers}")

            # Try to use the last conv layer if specified one doesn't exist
            if conv_layers:
                self.conv_layer = model.get_layer(conv_layers[-1])
                print(f"   Using alternative layer: '{self.conv_layer.name}
            else:
                raise

        # Create a functional model for gradient computation
        self._create_grad_model()

        print(f"✓ {model_name}: Using Grad-CAM layer '{self.conv_layer.name}

    def _create_grad_model(self):
        """Create the gradient model once (not in compute_heatmap)"""
        try:
            # Get input shape - handle different model types
            if hasattr(self.model, 'input_shape'):
                input_shape = self.model.input_shape[1:]
            elif hasattr(self.model, 'input'):
                # For functional models
                input_shape = self.model.input.shape[1:]
            else:
                # Default shape
                input_shape = (150, 150, 3)

            # Create input tensor
            self.grad_model_input = tf.keras.Input(shape=input_shape)

            # Run through the model up to the conv layer and final output
            x = self.grad_model_input
            conv_output = None

            # For Sequential models
            if isinstance(self.model, tf.keras.Sequential):
                for layer in self.model.layers:
                    x = layer(x)
                    if layer.name == self.conv_layer.name:
                        conv_output = x
            # For Functional models (like ResNet)
            else:
                # Get the layer output directly
                layer_output = self.model.get_layer(self.conv_layer.name).ou
                final_output = self.model.output
```

```
                # Create a model that outputs both
                self.grad_model = tf.keras.Model(
                    inputs=self.model.input,
                    outputs=[layer_output, final_output]
                )
                return

            # Create the gradient model for Sequential
            self.grad_model = tf.keras.Model(
                inputs=[self.grad_model_input],
                outputs=[conv_output, x]  # conv_output and final output
            )

        except Exception as e:
            print(f"❌ Error creating grad model for {self.model_name}: {e}"
            raise

    def compute_heatmap(self, image, class_index=None):
        """Compute Grad-CAM heatmap"""
        # Ensure image has batch dimension
        if len(image.shape) == 3:
            image = tf.expand_dims(image, axis=0)

        try:
            with tf.GradientTape() as tape:
                conv_outputs, predictions = self.grad_model(image, training=

                # Handle binary classification
                if predictions.shape[-1] == 1:
                    loss = predictions[:, 0]
                else:
                    if class_index is None:
                        class_index = tf.argmax(predictions[0])
                    loss = predictions[:, class_index]

            # Compute gradients
            grads = tape.gradient(loss, conv_outputs)

            # Global average pooling
            pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

            # Weight channels by gradients
            conv_outputs = conv_outputs[0]
            heatmap = tf.reduce_sum(conv_outputs * pooled_grads, axis=-1)

            # ReLU and normalize
            heatmap = tf.maximum(heatmap, 0)
            heatmap = heatmap.numpy()

            heatmap_min = np.min(heatmap)
            heatmap_max = np.max(heatmap)
            if heatmap_max > heatmap_min:
                heatmap = (heatmap - heatmap_min) / (heatmap_max - heatmap_m
            else:
                heatmap = np.zeros_like(heatmap)

            return heatmap

        except Exception as e:
            print(f"❌ Error computing heatmap for {self.model_name}: {e}")
            # Return a zero heatmap as fallback
            return np.zeros((image.shape[1] // 8, image.shape[2] // 8))
```

```
class EnhancedGradCAM(GradCAM):
    """Enhanced Grad-CAM specifically for Model 3 (VGG16) - Better pneumonia

    def compute_heatmap(self, image, class_index=None, use_guided=True, smoo
        """
        Enhanced Grad-CAM with improvements for pneumonia detection

        Args:
            image: Input image
            class_index: Target class (0=Normal, 1=Pneumonia)
            use_guided: Apply guided backpropagation for sharper heatmaps
            smooth_iterations: Number of smoothing iterations to reduce nois
        """
        # Ensure image has batch dimension
        if len(image.shape) == 3:
            image = tf.expand_dims(image, axis=0)

        try:
            with tf.GradientTape() as tape:
                conv_outputs, predictions = self.grad_model(image, training=

                # Handle binary classification
                if predictions.shape[-1] == 1:
                    loss = predictions[:, 0]
                else:
                    if class_index is None:
                        class_index = tf.argmax(predictions[0])
                    loss = predictions[:, class_index]

            # Compute gradients
            grads = tape.gradient(loss, conv_outputs)

            # IMPROVEMENT 1: Guided Backpropagation (suppress negative gradi
            if use_guided:
                grads = tf.cast(conv_outputs > 0, 'float32') * tf.cast(grads

            # IMPROVEMENT 2: Use ReLU on gradients before pooling
            grads = tf.maximum(grads, 0)

            # Global average pooling on gradients
            pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))

            # Weight channels by gradients
            conv_outputs = conv_outputs[0]
            heatmap = tf.reduce_sum(conv_outputs * pooled_grads, axis=-1)

            # Apply ReLU
            heatmap = tf.maximum(heatmap, 0)
            heatmap = heatmap.numpy()

            # IMPROVEMENT 3: Smooth the heatmap to reduce noise
            for _ in range(smooth_iterations):
                heatmap = cv2.GaussianBlur(heatmap, (3, 3), 0)

            # IMPROVEMENT 4: Enhanced normalization with percentile clipping
            # This removes outliers and improves contrast
            lower_percentile = np.percentile(heatmap, 10)
            upper_percentile = np.percentile(heatmap, 99)
            heatmap = np.clip(heatmap, lower_percentile, upper_percentile)

            # Normalize to [0, 1]
            heatmap_min = np.min(heatmap)
            heatmap_max = np.max(heatmap)
            if heatmap_max > heatmap_min:
```

```
                heatmap = (heatmap - heatmap_min) / (heatmap_max - heatmap_m
            else:
                heatmap = np.zeros_like(heatmap)

            # IMPROVEMENT 5: Apply power transformation to enhance high-acti
            # This makes the important areas more prominent
            heatmap = np.power(heatmap, 0.7)  # Gamma correction

            return heatmap

        except Exception as e:
            print(f"❌ Error computing heatmap for {self.model_name}: {e}")
            return np.zeros((image.shape[1] // 8, image.shape[2] // 8))


# ============================================
# STEP 2: FIND LAST CONV LAYER FOR EACH MODEL
# ============================================

def find_last_conv_layer(model):
    """Find the last convolutional layer in a model"""
    conv_layers = []
    for layer in model.layers:
        if 'conv' in layer.name.lower():
            conv_layers.append(layer.name)

    if len(conv_layers) == 0:
        raise ValueError("No convolutional layers found in model")

    return conv_layers[-1]

print("\n" + "="*60)
print("IDENTIFYING CONVOLUTIONAL LAYERS")
print("="*60)

# Check if models exist before trying to find layers
print("\nChecking model availability...")
print(f"model1 exists: {'model1' in dir()}")
print(f"model2 exists: {'model2' in dir()}")
print(f"model3 exists: {'model3' in dir()}")

# Find last conv layer for each model (only if models exist)
if 'model1' in dir():
    try:
        last_conv_model1 = find_last_conv_layer(model1)
        print(f"\n✓ Model 1 (Simple CNN):  {last_conv_model1}")
    except Exception as e:
        print(f"\n✗ Error finding conv layer for Model 1: {e}")
        last_conv_model1 = None

if 'model2' in dir():
    try:
        last_conv_model2 = find_last_conv_layer(model2)
        print(f"✓ Model 2 (ResNet):      {last_conv_model2}")
    except Exception as e:
        print(f"✗ Error finding conv layer for Model 2: {e}")
        last_conv_model2 = None

if 'model3' in dir():
    try:
        last_conv_model3 = find_last_conv_layer(model3)
        print(f"✓ Model 3 (VGG16):       {last_conv_model3}")
    except Exception as e:
        print(f"✗ Error finding conv layer for Model 3: {e}")
```

```
        last_conv_model3 = None

# ============================================
# STEP 3: INITIALIZE GRAD-CAM FOR ALL MODELS
# ============================================

print("\n" + "="*60)
print("INITIALIZING GRAD-CAM")
print("="*60)

# Initialize Grad-CAM objects only for models that exist
gradcam_model1 = None
gradcam_model2 = None
gradcam_model3 = None

# For Model 1 - Standard Grad-CAM
if 'model1' in dir() and last_conv_model1 is not None:
    try:
        print(f"\nCreating Grad-CAM for Model 1 (Simple CNN)...")

        # Ensure model1 is built and called
        if not model1.built:
            model1.build(input_shape=(None, 150, 150, 3))
            print(f"  Built Model 1")

        # Create dummy call to initialize model
        dummy_input = tf.zeros((1, 150, 150, 3))
        _ = model1(dummy_input)
        print(f"  Initialized Model 1")

        # Create Standard Grad-CAM
        gradcam_model1 = GradCAM(model1, layer_name=last_conv_model1, model_
        print(f"  ✓ Grad-CAM created for Model 1")
    except Exception as e:
        print(f"  ✗ Error creating Grad-CAM for Model 1: {e}")

# For Model 2 - Standard Grad-CAM
if 'model2' in dir() and last_conv_model2 is not None:
    try:
        print(f"\nCreating Grad-CAM for Model 2 (ResNet)...")

        # Ensure model2 is built
        if hasattr(model2, 'built') and not model2.built:
            model2.build(input_shape=(None, 150, 150, 3))
            print(f"  Built Model 2")

        # Create Standard Grad-CAM
        gradcam_model2 = GradCAM(model2, layer_name=last_conv_model2, model_
        print(f"  ✓ Grad-CAM created for Model 2")
    except Exception as e:
        print(f"  ✗ Error creating Grad-CAM for Model 2: {e}")

# For Model 3 - ENHANCED Grad-CAM
if 'model3' in dir() and last_conv_model3 is not None:
    try:
        print(f"\nCreating ENHANCED Grad-CAM for Model 3 (VGG16)...")

        # Ensure model3 is built
        if hasattr(model3, 'built') and not model3.built:
            model3.build(input_shape=(None, 150, 150, 3))
            print(f"  Built Model 3")

        # Create ENHANCED Grad-CAM for Model 3
        gradcam_model3 = EnhancedGradCAM(model3, layer_name=last_conv_model3
```

```
        print(f"  ✓ ENHANCED Grad-CAM created for Model 3")
        print(f"  → Using guided backpropagation, smoothing, and contrast en
    except Exception as e:
        print(f"  ✗ Error creating Grad-CAM for Model 3: {e}")

print("\n" + "="*60)
print("GRAD-CAM INITIALIZATION SUMMARY")
print("="*60)
print(f"Model 1 Grad-CAM: {'✓ Ready (Standard)' if gradcam_model1 else '✗ No
print(f"Model 2 Grad-CAM: {'✓ Ready (Standard)' if gradcam_model2 else '✗ No
print(f"Model 3 Grad-CAM: {'✓ Ready (ENHANCED)' if gradcam_model3 else '✗ No

# ============================================
# STEP 4: ENHANCED VISUALIZATION FUNCTION
# ============================================

def create_overlay(img, heatmap):
    """Create overlay of heatmap on original image"""
    # Resize heatmap to match image size
    heatmap_resized = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

    # Convert heatmap to RGB
    heatmap_colored = cv2.applyColorMap(np.uint8(255 * heatmap_resized), cv2
    heatmap_colored = cv2.cvtColor(heatmap_colored, cv2.COLOR_BGR2RGB)

    # Normalize image to 0-255
    img_normalized = (img * 255).astype(np.uint8)
    if len(img_normalized.shape) == 2:
        img_normalized = cv2.cvtColor(img_normalized, cv2.COLOR_GRAY2RGB)

    # Create overlay (60% image, 40% heatmap)
    overlay = cv2.addWeighted(img_normalized, 0.6, heatmap_colored, 0.4, 0)

    return overlay

def visualize_gradcam_triple(img, true_label, pred_label, pred_prob, gradcam
    """Visualize Grad-CAM: Heatmap, Original, Overlay"""
    # Compute heatmap
    heatmap = gradcam.compute_heatmap(img, class_index=pred_label)
    heatmap_resized = cv2.resize(heatmap, (img.shape[1], img.shape[0]))

    # Create overlay
    overlay = create_overlay(img, heatmap)

    # Create figure with 3 subplots
    fig = plt.figure(figsize=(15, 5))

    # 1. Heatmap
    plt.subplot(1, 3, 1)
    plt.title("Grad-CAM Heatmap", fontsize=12, fontweight='bold')
    plt.axis("off")
    plt.imshow(heatmap_resized, cmap='jet', interpolation='bilinear')

    # 2. Original Image
    plt.subplot(1, 3, 2)
    plt.title("Original Image", fontsize=12, fontweight='bold')
    plt.axis("off")
    if len(img.shape) == 2:
        plt.imshow(img, cmap='gray', interpolation='bilinear')
    else:
        plt.imshow(img, interpolation='bilinear')

    # 3. Overlay
    plt.subplot(1, 3, 3)
```

```
    plt.title("Heatmap Overlay", fontsize=12, fontweight='bold')
    plt.axis("off")
    plt.imshow(overlay, interpolation='bilinear')

    # Create title
    true_class = 'NORMAL' if true_label == 0 else 'PNEUMONIA'
    pred_class = 'NORMAL' if pred_label == 0 else 'PNEUMONIA'
    confidence = pred_prob if pred_label == 1 else (1 - pred_prob)
    match_symbol = "✓" if true_label == pred_label else "✗"

    if image_id is not None:
        title = f"{model_name} | Image #{image_id} | {match_symbol} True: {t
    else:
        title = f"{model_name} | {match_symbol} True: {true_class} | Pred: -

    fig.suptitle(title, fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

# ============================================
# STEP 5: VISUALIZE ALL SCENARIOS FOR ALL MODELS
# ============================================

def analyze_model_gradcam(model_name, gradcam, y_pred, y_pred_prob, num_samp
    """Analyze one model with Grad-CAM for all 4 scenarios"""

    if gradcam is None:
        print(f"\n❌ {model_name}: Grad-CAM not available. Skipping analysis
        return

    print("\n" + "="*60)
    print(f"GRAD-CAM ANALYSIS: {model_name}")
    print("="*60)

    # Define scenarios
    scenarios = [
        {
            'name': 'SCENARIO 1: True Negative (TN) - Normal→Normal ✓',
            'mask': (y_test == 0) & (y_pred == 0),
            'short': 'TN'
        },
        {
            'name': 'SCENARIO 2: True Positive (TP) - Pneumonia→Pneumonia ✓
            'mask': (y_test == 1) & (y_pred == 1),
            'short': 'TP'
        },
        {
            'name': 'SCENARIO 3: False Positive (FP) - Normal→Pneumonia ✗',
            'mask': (y_test == 0) & (y_pred == 1),
            'short': 'FP'
        },
        {
            'name': 'SCENARIO 4: False Negative (FN) - Pneumonia→Normal ✗',
            'mask': (y_test == 1) & (y_pred == 0),
            'short': 'FN'
        }
    ]

    # Visualize each scenario
    np.random.seed(42)
    for scenario in scenarios:
        indices = np.where(scenario['mask'])[0]

        if len(indices) == 0:
```

```
            print(f"\n❌ {scenario['name']}: No cases found")
            continue

        print(f"\n{'='*60}")
        print(f"{scenario['name']}")
        print(f"{'='*60}")
        print(f"Found {len(indices)} cases, showing {min(num_samples, len(in

        # Select samples
        selected_indices = np.random.choice(indices, min(num_samples, len(in

        for i, idx in enumerate(selected_indices, 1):
            img = X_test_scaled[idx]
            true_label = y_test[idx]
            pred_prob = y_pred_prob[idx][0] if isinstance(y_pred_prob[idx],
            pred_label = y_pred[idx]

            print(f"Sample {i}/{len(selected_indices)} - Image Index: {idx}"
            visualize_gradcam_triple(img, true_label, pred_label, pred_prob,

    # Print summary
    print(f"\n{'='*60}")
    print(f"{model_name} - PREDICTION SUMMARY")
    print(f"{'='*60}")

    tn = np.sum((y_test == 0) & (y_pred == 0))
    tp = np.sum((y_test == 1) & (y_pred == 1))
    fp = np.sum((y_test == 0) & (y_pred == 1))
    fn = np.sum((y_test == 1) & (y_pred == 0))
    total = len(y_test)

    print(f"\nTest Set Breakdown (Total: {total} samples):")
    print(f"{'='*60}")
    print(f"✓ True Negative  (TN):  {tn:4d} ({tn/total*100:5.2f}%)")
    print(f"✓ True Positive  (TP):  {tp:4d} ({tp/total*100:5.2f}%)")
    print(f"✗ False Positive (FP):  {fp:4d} ({fp/total*100:5.2f}%)")
    print(f"✗ False Negative (FN):  {fn:4d} ({fn/total*100:5.2f}%)")
    print(f"{'='*60}")
    print(f"Overall Accuracy: {(tn + tp)/total*100:.2f}%")

# ============================================
# STEP 6: RUN ANALYSIS FOR ALL 3 MODELS
# ============================================

print("\n" + "="*60)
print("STARTING GRAD-CAM ANALYSIS FOR ALL MODELS")
print("Generating 10 samples per scenario (TN, TP, FP, FN)")
print("Model 3 uses ENHANCED heatmap generation")
print("="*60)

# Check if y_pred variables exist
print("\nChecking prediction variables...")
print(f"y_pred1 exists: {'y_pred1' in dir()}")
print(f"y_pred2 exists: {'y_pred2' in dir()}")
print(f"y_pred3 exists: {'y_pred3' in dir()}")
print(f"y_pred1_prob exists: {'y_pred1_prob' in dir()}")
print(f"y_pred2_prob exists: {'y_pred2_prob' in dir()}")
print(f"y_pred3_prob exists: {'y_pred3_prob' in dir()}")

# Model 1: Simple CNN (only if everything is available)
if gradcam_model1 and 'y_pred1' in dir() and 'y_pred1_prob' in dir():
    analyze_model_gradcam(
        model_name="Model 1 (Simple CNN)",
        gradcam=gradcam_model1,
```

```
        y_pred=y_pred1,
        y_pred_prob=y_pred1_prob,
        num_samples=10
    )
else:
    print("\n❌ Skipping Model 1: Missing Grad-CAM or predictions")

# Model 2: ResNet (only if everything is available)
if gradcam_model2 and 'y_pred2' in dir() and 'y_pred2_prob' in dir():
    analyze_model_gradcam(
        model_name="Model 2 (ResNet)",
        gradcam=gradcam_model2,
        y_pred=y_pred2,
        y_pred_prob=y_pred2_prob,
        num_samples=10
    )
else:
    print("\n❌ Skipping Model 2: Missing Grad-CAM or predictions")

# Model 3: VGG16 with ENHANCED Grad-CAM (only if everything is available)
if gradcam_model3 and 'y_pred3' in dir() and 'y_pred3_prob' in dir():
    analyze_model_gradcam(
        model_name="Model 3 (VGG16) - ENHANCED",
        gradcam=gradcam_model3,
        y_pred=y_pred3,
        y_pred_prob=y_pred3_prob,
        num_samples=10
    )
else:
    print("\n❌ Skipping Model 3: Missing Grad-CAM or predictions")

print("\n" + "="*60)
print("GRAD-CAM ANALYSIS COMPLETE!")
print("="*60)
print("\nMODEL 3 ENHANCEMENTS:")
print("  ✓ Guided Backpropagation - Sharper heatmaps")
print("  ✓ Gaussian Smoothing - Reduced noise")
print("  ✓ Percentile Normalization - Better contrast")
print("  ✓ Gamma Correction - Enhanced important regions")
print("="*60)
```

```
============================================================
GRAD-CAM ANALYSIS - ALL 3 MODELS
Enhanced heatmap for Model 3 (VGG16)
============================================================

============================================================
IDENTIFYING CONVOLUTIONAL LAYERS
============================================================

Checking model availability...
model1 exists: True
model2 exists: True
model3 exists: True

✓ Model 1 (Simple CNN):  conv2d_2
✓ Model 2 (ResNet):      block3b_conv2
✓ Model 3 (VGG16):       conv2d_9

============================================================
INITIALIZING GRAD-CAM
============================================================

Creating Grad-CAM for Model 1 (Simple CNN)...
  Initialized Model 1
✓ Model 1 (Simple CNN): Using Grad-CAM layer 'conv2d_2'
  ✓ Grad-CAM created for Model 1

Creating Grad-CAM for Model 2 (ResNet)...
✓ Model 2 (ResNet): Using Grad-CAM layer 'block3b_conv2'
  ✓ Grad-CAM created for Model 2

Creating ENHANCED Grad-CAM for Model 3 (VGG16)...
✓ Model 3 (VGG16) - Enhanced: Using Grad-CAM layer 'conv2d_9'
  ✓ ENHANCED Grad-CAM created for Model 3
  → Using guided backpropagation, smoothing, and contrast enhancement

============================================================
GRAD-CAM INITIALIZATION SUMMARY
============================================================
Model 1 Grad-CAM: ✓ Ready (Standard)
Model 2 Grad-CAM: ✓ Ready (Standard)
Model 3 Grad-CAM: ✓ Ready (ENHANCED)

============================================================
STARTING GRAD-CAM ANALYSIS FOR ALL MODELS
Generating 10 samples per scenario (TN, TP, FP, FN)
Model 3 uses ENHANCED heatmap generation
============================================================

Checking prediction variables...
y_pred1 exists: True
y_pred2 exists: True
y_pred3 exists: True
y_pred1_prob exists: True
y_pred2_prob exists: True
y_pred3_prob exists: True

============================================================
GRAD-CAM ANALYSIS: Model 1 (Simple CNN)
============================================================

============================================================
SCENARIO 1: True Negative (TN) - Normal→Normal ✓
============================================================
```

```
Found 115 cases, showing 10 examples...

Sample 1/10 - Image Index: 170
```


```
Sample 2/10 - Image Index: 9
```

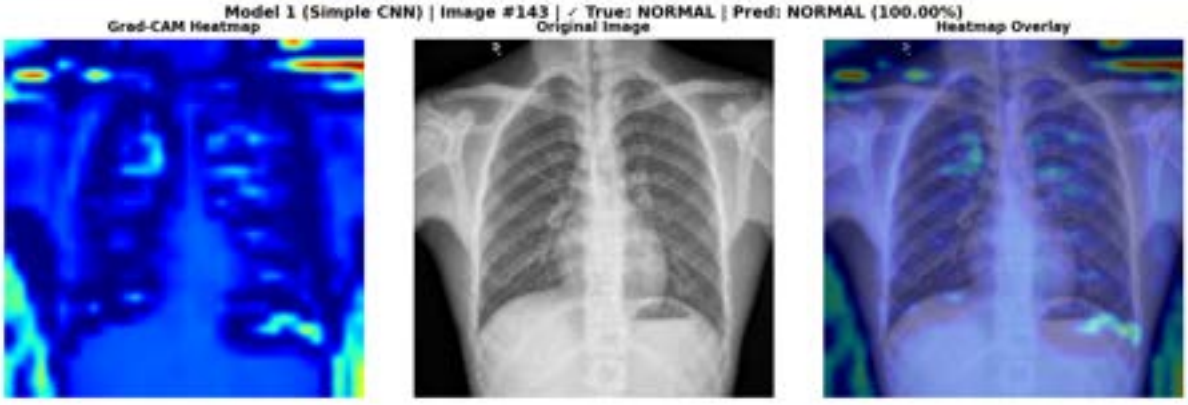

```
Sample 3/10 - Image Index: 74
```


```
Sample 4/10 - Image Index: 141
```


```
Sample 5/10 - Image Index: 19
```

Sample 6/10 – Image Index: 82

Sample 7/10 – Image Index: 143



Sample 8/10 – Image Index: 137

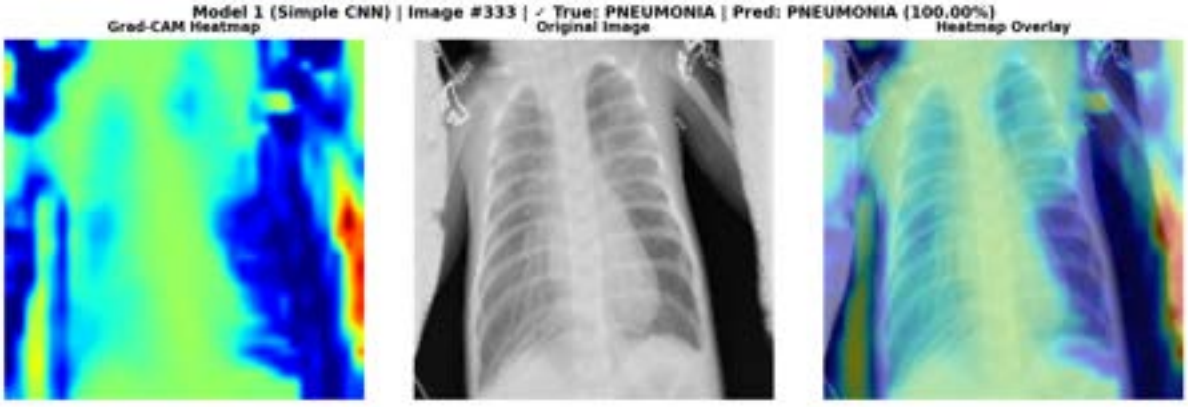
Sample 9/10 – Image Index: 89

Sample 10/10 – Image Index: 20

==
SCENARIO 2: True Positive (TP) – Pneumonia→Pneumonia ✓
==
Found 387 cases, showing 10 examples...

Sample 1/10 – Image Index: 539

Sample 2/10 – Image Index: 333



Sample 3/10 – Image Index: 498

Sample 4/10 — Image Index: 387

Sample 5/10 — Image Index: 321

Sample 6/10 — Image Index: 365

Sample 7/10 — Image Index: 438

Sample 8/10 – Image Index: 305

Sample 9/10 – Image Index: 561

Sample 10/10 – Image Index: 482


```
============================================================
SCENARIO 3: False Positive (FP) – Normal→Pneumonia ✗
============================================================
Found 119 cases, showing 10 examples...

Sample 1/10 – Image Index: 43
```

Sample 2/10 – Image Index: 27

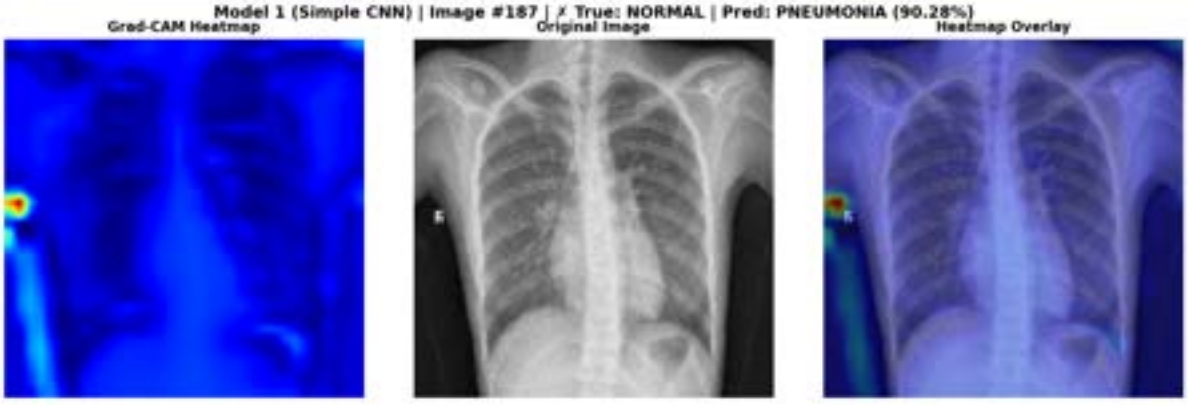
Sample 3/10 – Image Index: 217

Sample 4/10 – Image Index: 35

Sample 5/10 – Image Index: 196

Sample 6/10 – Image Index: 100

Sample 7/10 – Image Index: 187



Sample 8/10 – Image Index: 84

Sample 9/10 – Image Index: 56

Sample 10/10 – Image Index: 87

==
SCENARIO 4: False Negative (FN) – Pneumonia→Normal ✗
==
Found 3 cases, showing 3 examples...

Sample 1/3 – Image Index: 584

Sample 2/3 – Image Index: 547

Sample 3/3 – Image Index: 462


```
============================================================
Model 1 (Simple CNN) - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):   115 (18.43%)
✓ True Positive  (TP):   387 (62.02%)
✗ False Positive (FP):   119 (19.07%)
✗ False Negative (FN):     3 ( 0.48%)
============================================================
Overall Accuracy: 80.45%

============================================================
GRAD-CAM ANALYSIS: Model 2 (ResNet)
============================================================

============================================================
SCENARIO 1: True Negative (TN) - Normal→Normal ✓
============================================================
Found 105 cases, showing 10 examples...

Sample 1/10 - Image Index: 62
```


```
Sample 2/10 - Image Index: 140
```


```
Sample 3/10 - Image Index: 137
```

Sample 4/10 — Image Index: 116

Sample 5/10 — Image Index: 85

Sample 6/10 — Image Index: 202

Sample 7/10 — Image Index: 233

Sample 8/10 – Image Index: 96

Sample 9/10 – Image Index: 17

Sample 10/10 – Image Index: 2


```
============================================================
SCENARIO 2: True Positive (TP) – Pneumonia→Pneumonia ✓
============================================================
Found 386 cases, showing 10 examples...

Sample 1/10 – Image Index: 360
```

Sample 2/10 – Image Index: 319

Sample 3/10 – Image Index: 397

Sample 4/10 – Image Index: 491

Sample 5/10 – Image Index: 303

Sample 6/10 – Image Index: 409

Sample 7/10 – Image Index: 520

Sample 8/10 – Image Index: 423

Sample 9/10 – Image Index: 389

Sample 10/10 – Image Index: 391

==
SCENARIO 3: False Positive (FP) – Normal→Pneumonia ✗
==
Found 129 cases, showing 10 examples...

Sample 1/10 – Image Index: 132

Sample 2/10 – Image Index: 192

Sample 3/10 – Image Index: 228

Sample 4/10 – Image Index: 206

Sample 5/10 – Image Index: 148

Sample 6/10 – Image Index: 94

Sample 7/10 – Image Index: 223

Sample 8/10 – Image Index: 226

Sample 9/10 – Image Index: 99

Sample 10/10 – Image Index: 93


```
============================================================
SCENARIO 4: False Negative (FN) – Pneumonia→Normal ✗
============================================================
Found 4 cases, showing 4 examples...

Sample 1/4 – Image Index: 245
```

Sample 2/4 – Image Index: 547

Sample 3/4 – Image Index: 584

Sample 4/4 – Image Index: 336

```
============================================================
Model 2 (ResNet) - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):   105 (16.83%)
✓ True Positive  (TP):   386 (61.86%)
✗ False Positive (FP):   129 (20.67%)
✗ False Negative (FN):     4 ( 0.64%)
============================================================
Overall Accuracy: 78.69%

============================================================
GRAD-CAM ANALYSIS: Model 3 (VGG16) - ENHANCED
============================================================

============================================================
SCENARIO 1: True Negative (TN) - Normal→Normal ✓
============================================================
Found 81 cases, showing 10 examples...

Sample 1/10 - Image Index: 71
```

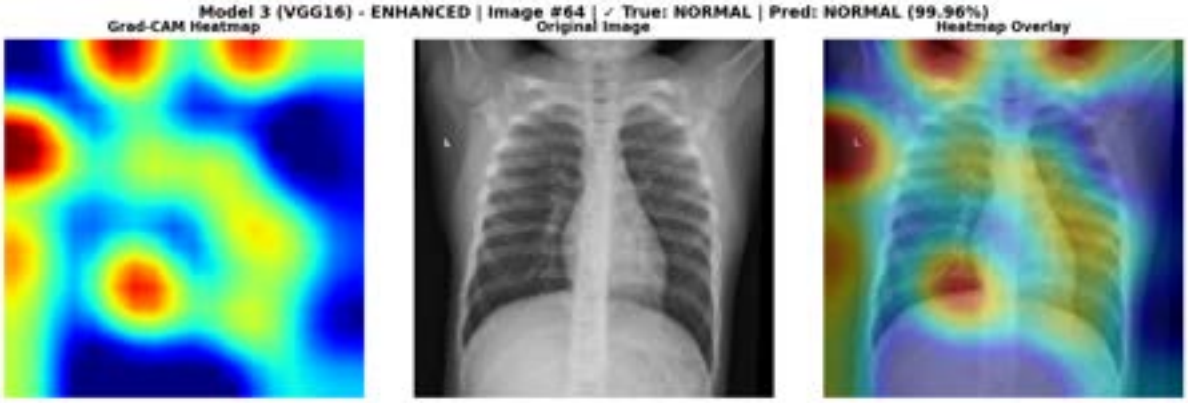

```
Sample 2/10 - Image Index: 2
```


```
Sample 3/10 - Image Index: 54
```


```
Sample 4/10 - Image Index: 72
```

Sample 5/10 – Image Index: 39

Sample 6/10 – Image Index: 64



Sample 7/10 – Image Index: 19

Sample 8/10 – Image Index: 204

Sample 9/10 – Image Index: 10

Sample 10/10 – Image Index: 23


```
============================================================
SCENARIO 2: True Positive (TP) – Pneumonia→Pneumonia ✓
============================================================
Found 388 cases, showing 10 examples...

Sample 1/10 – Image Index: 292
```

Sample 2/10 – Image Index: 307

Sample 3/10 – Image Index: 545

Sample 4/10 – Image Index: 410

Sample 5/10 – Image Index: 567

Sample 6/10 – Image Index: 439

Sample 7/10 — Image Index: 390

Sample 8/10 — Image Index: 289

Sample 9/10 — Image Index: 555

Sample 10/10 — Image Index: 243

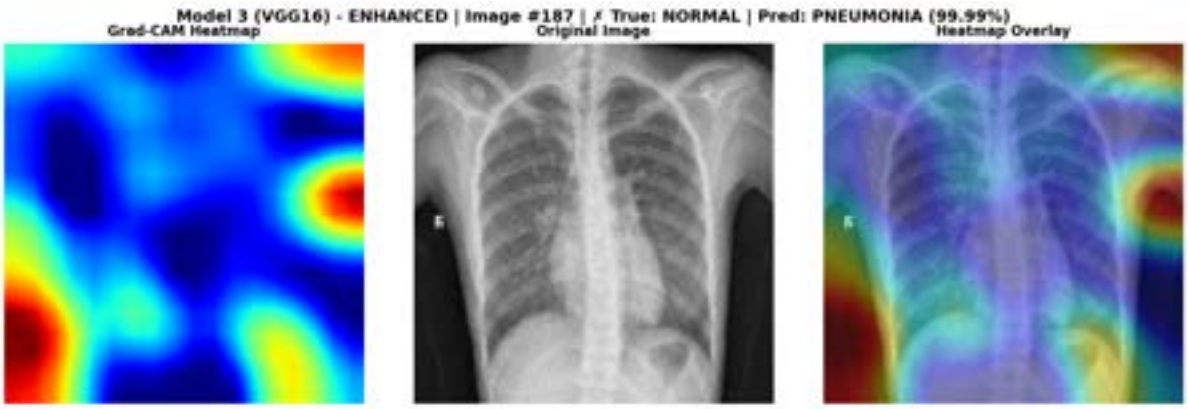

```
============================================================
SCENARIO 3: False Positive (FP) - Normal→Pneumonia ✗
============================================================
Found 153 cases, showing 10 examples...

Sample 1/10 - Image Index: 224
```


```
Sample 2/10 - Image Index: 153
```


```
Sample 3/10 - Image Index: 102
```


```
Sample 4/10 - Image Index: 192
```

Sample 5/10 – Image Index: 187



Sample 6/10 – Image Index: 182

Sample 7/10 – Image Index: 93

Sample 8/10 – Image Index: 148

Sample 9/10 – Image Index: 219

Sample 10/10 – Image Index: 99

==

SCENARIO 4: False Negative (FN) – Pneumonia→Normal ✗

==

Found 2 cases, showing 2 examples...

Sample 1/2 – Image Index: 321

Sample 2/2 – Image Index: 584


```
============================================================
Model 3 (VGG16) - ENHANCED - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):    81 (12.98%)
✓ True Positive  (TP):   388 (62.18%)
✗ False Positive (FP):   153 (24.52%)
✗ False Negative (FN):     2 ( 0.32%)
============================================================
Overall Accuracy: 75.16%

============================================================
GRAD-CAM ANALYSIS COMPLETE!
============================================================

MODEL 3 ENHANCEMENTS:
  ✓ Guided Backpropagation - Sharper heatmaps
  ✓ Gaussian Smoothing - Reduced noise
  ✓ Percentile Normalization - Better contrast
  ✓ Gamma Correction - Enhanced important regions
============================================================
```

In [11]:

```
# ============================================
# RISE (Randomized Input Sampling for Explanation)
# FOR ALL 3 MODELS - IMPROVED VERSION
# Clean heatmaps with proper smoothing
# ============================================

import numpy as np
import matplotlib.pyplot as plt
import cv2
from tqdm import tqdm

print("\n" + "="*60)
print("RISE - EXPLAINABLE AI FOR ALL 3 MODELS")
print("Improved Implementation with Smooth Heatmaps")
print("="*60)

# ============================================
# STEP 1: IMPROVED RISE IMPLEMENTATION
# ============================================

class RISEExplainer:
    """
    RISE: Randomized Input Sampling for Explanation
    Improved version with smoother heatmaps
    """

    def __init__(self, model, model_name="Model", input_size=(150, 150),
```

```
                 num_masks=2000, mask_prob=0.5):
        """
        Initialize RISE explainer

        Args:
            model: Trained Keras model
            model_name: Name for display
            input_size: Size of input images (height, width)
            num_masks: Number of random masks to generate
            mask_prob: Probability of each cell being 1 in the mask
        """
        self.model = model
        self.model_name = model_name
        self.input_size = input_size
        self.num_masks = num_masks
        self.mask_prob = mask_prob
        print(f"✓ {model_name}: RISE initialized with {num_masks} masks")

    def generate_masks(self, num_masks, mask_size=(7, 7)):
        """
        Generate random binary masks with smooth upsampling
        Key improvement: Using INTER_LINEAR for smoother masks
        """
        masks = []
        for _ in range(num_masks):
            # Generate small random mask
            mask = np.random.rand(*mask_size) < self.mask_prob

            # Upscale to input size with LINEAR interpolation (smoother)
            mask_upscaled = cv2.resize(
                mask.astype(float),
                self.input_size,
                interpolation=cv2.INTER_LINEAR
            )
            masks.append(mask_upscaled)

        return np.array(masks)

    def explain(self, image, target_class=1, batch_size=50):
        """
        Generate RISE explanation for an image

        Args:
            image: Input image (normalized, shape: H x W x C)
            target_class: Class to explain (0=Normal, 1=Pneumonia)
            batch_size: Process masks in batches for efficiency

        Returns:
            saliency_map: Smoothed importance map
        """
        # Generate random masks
        masks = self.generate_masks(self.num_masks)

        # Initialize saliency map
        saliency_map = np.zeros(self.input_size)

        # Process in batches for efficiency
        num_batches = int(np.ceil(self.num_masks / batch_size))

        for batch_idx in tqdm(range(num_batches),
                              desc=f"  Computing RISE",
                              leave=False):
            start_idx = batch_idx * batch_size
            end_idx = min((batch_idx + 1) * batch_size, self.num_masks)
```

```
            batch_masks = masks[start_idx:end_idx]

            # Create batch of masked images
            batch_images = []
            for mask in batch_masks:
                masked_image = image * mask[:, :, np.newaxis]
                batch_images.append(masked_image)

            batch_images = np.array(batch_images)

            # Get predictions for batch
            batch_preds = self.model.predict(batch_images, verbose=0)

            # Handle different output formats
            if batch_preds.shape[-1] == 1:
                batch_preds = batch_preds.flatten()
            else:
                batch_preds = batch_preds[:, target_class]

            # Accumulate weighted masks
            for mask, pred in zip(batch_masks, batch_preds):
                if target_class == 1:
                    saliency_map += mask * pred
                else:
                    saliency_map += mask * (1 - pred)

        # Normalize saliency map
        saliency_map = saliency_map / self.num_masks

        # KEY IMPROVEMENT: Apply Gaussian smoothing to reduce patchiness
        saliency_map = cv2.GaussianBlur(saliency_map, (15, 15), 0)

        # Normalize to [0, 1]
        saliency_min = np.min(saliency_map)
        saliency_max = np.max(saliency_map)
        if saliency_max > saliency_min:
            saliency_map = (saliency_map - saliency_min) / (saliency_max - s

        return saliency_map

# ============================================
# STEP 2: INITIALIZE RISE FOR ALL 3 MODELS
# ============================================

print("\n" + "="*60)
print("INITIALIZING RISE FOR ALL MODELS")
print("="*60)

# Initialize RISE objects
rise_model1 = None
rise_model2 = None
rise_model3 = None

# Model 1: Simple CNN
if 'model1' in dir():
    try:
        print("\nModel 1 (Simple CNN):")
        rise_model1 = RISEExplainer(
            model=model1,
            model_name="Model 1 (Simple CNN)",
            input_size=(150, 150),
            num_masks=2000,
            mask_prob=0.5
        )
```

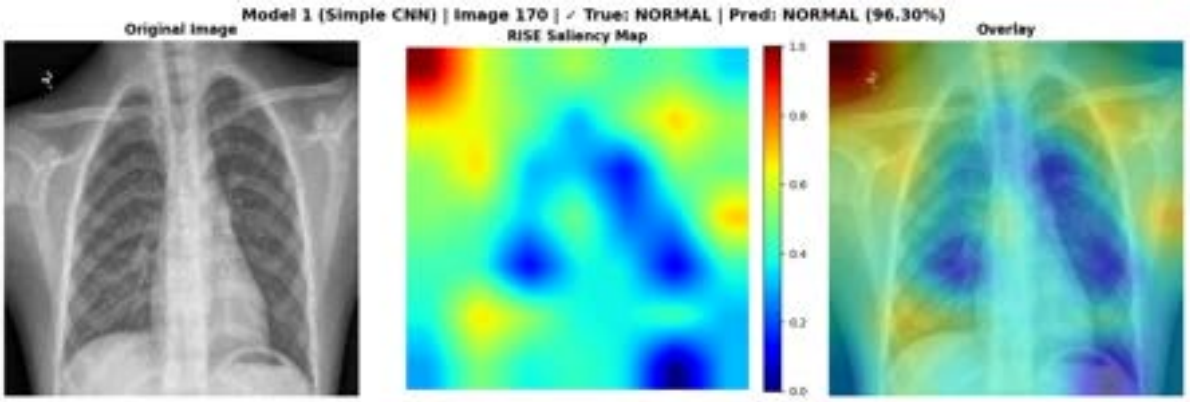
```
    except Exception as e:
        print(f"  ✗ Error: {e}")

# Model 2: ResNet
if 'model2' in dir():
    try:
        print("\nModel 2 (ResNet):")
        rise_model2 = RISEExplainer(
            model=model2,
            model_name="Model 2 (ResNet)",
            input_size=(150, 150),
            num_masks=2000,
            mask_prob=0.5
        )
    except Exception as e:
        print(f"  ✗ Error: {e}")

# Model 3: VGG16 (if exists)
if 'model3' in dir():
    try:
        print("\nModel 3 (VGG16):")
        rise_model3 = RISEExplainer(
            model=model3,
            model_name="Model 3 (VGG16)",
            input_size=(150, 150),
            num_masks=2000,
            mask_prob=0.5
        )
    except Exception as e:
        print(f"  ✗ Error: {e}")

print("\n" + "="*60)
print("RISE INITIALIZATION COMPLETE")
print("="*60)

# ============================================
# STEP 3: VISUALIZATION FUNCTION
# ============================================

def plot_rise_explanation(image, saliency_map, true_label, pred_label,
                          pred_prob, model_name, image_id=None):
    """
    Plot RISE explanation with 3 panels: Original, Heatmap, Overlay
    """
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # 1. Original image
    axes[0].imshow(image, cmap='gray')
    axes[0].set_title('Original Image', fontsize=12, fontweight='bold')
    axes[0].axis('off')

    # 2. RISE Saliency Map
    im = axes[1].imshow(saliency_map, cmap='jet', interpolation='bilinear')
    axes[1].set_title('RISE Saliency Map', fontsize=12, fontweight='bold')
    axes[1].axis('off')
    plt.colorbar(im, ax=axes[1], fraction=0.046, pad=0.04)

    # 3. Overlay
    axes[2].imshow(image, cmap='gray')
    axes[2].imshow(saliency_map, cmap='jet', alpha=0.5, interpolation='bili
    axes[2].set_title('Overlay', fontsize=12, fontweight='bold')
    axes[2].axis('off')

    # Create title
```

```
    true_class = 'NORMAL' if true_label == 0 else 'PNEUMONIA'
    pred_class = 'NORMAL' if pred_label == 0 else 'PNEUMONIA'
    confidence = pred_prob if pred_label == 1 else (1 - pred_prob)
    match_symbol = "✓" if true_label == pred_label else "✗"

    if image_id is not None:
        title = f"{model_name} | Image {image_id} | {match_symbol} True: {t
    else:
        title = f"{model_name} | {match_symbol} True: {true_class} | Pred: 

    fig.suptitle(title, fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

# ============================================
# STEP 4: ANALYZE EACH MODEL
# ============================================

def analyze_model_rise(model_name, rise_obj, y_pred, y_pred_prob, num_sample
    """
    Analyze one model with RISE for all 4 scenarios (TN, TP, FP, FN)
    Shows 10 samples per scenario
    """
    if rise_obj is None:
        print(f"\n❌ {model_name}: RISE not available")
        return

    print("\n" + "="*60)
    print(f"RISE ANALYSIS: {model_name}")
    print("="*60)

    # Define scenarios
    scenarios = [
        {
            'name': '✓ SCENARIO 1: True Negative (Normal → Normal)',
            'mask': (y_test == 0) & (y_pred == 0),
            'short': 'TN'
        },
        {
            'name': '✓ SCENARIO 2: True Positive (Pneumonia → Pneumonia)',
            'mask': (y_test == 1) & (y_pred == 1),
            'short': 'TP'
        },
        {
            'name': '✗ SCENARIO 3: False Positive (Normal → Pneumonia)',
            'mask': (y_test == 0) & (y_pred == 1),
            'short': 'FP'
        },
        {
            'name': '✗ SCENARIO 4: False Negative (Pneumonia → Normal)',
            'mask': (y_test == 1) & (y_pred == 0),
            'short': 'FN'
        }
    ]

    # Process each scenario
    np.random.seed(42)
    for scenario in scenarios:
        indices = np.where(scenario['mask'])[0]

        if len(indices) == 0:
            print(f"\n❌ {scenario['name']}: No cases found")
            continue
```

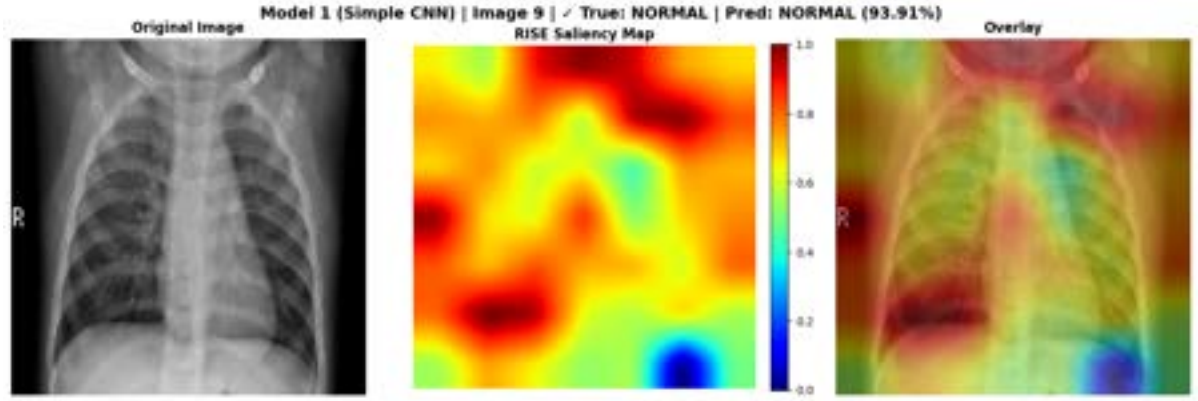
```
        print(f"\n{'='*60}")
        print(f"{scenario['name']}")
        print(f"{'='*60}")
        print(f"Found {len(indices)} cases, showing {min(num_samples, len(ir

        # Select samples
        selected_indices = np.random.choice(
            indices,
            min(num_samples, len(indices)),
            replace=False
        )

        # Generate RISE for each sample
        for i, idx in enumerate(selected_indices, 1):
            print(f"\n  Sample {i}/{len(selected_indices)} - Image {idx}")

            img = X_test_scaled[idx]
            true_label = y_test[idx]
            pred_prob = y_pred_prob[idx][0] if isinstance(y_pred_prob[idx],
            pred_label = y_pred[idx]

            # Generate RISE explanation
            saliency_map = rise_obj.explain(img, target_class=pred_label)

            # Visualize
            plot_rise_explanation(
                image=img,
                saliency_map=saliency_map,
                true_label=true_label,
                pred_label=pred_label,
                pred_prob=pred_prob,
                model_name=model_name,
                image_id=idx
            )

    # Print summary
    print(f"\n{'='*60}")
    print(f"{model_name} - PREDICTION SUMMARY")
    print(f"{'='*60}")

    tn = np.sum((y_test == 0) & (y_pred == 0))
    tp = np.sum((y_test == 1) & (y_pred == 1))
    fp = np.sum((y_test == 0) & (y_pred == 1))
    fn = np.sum((y_test == 1) & (y_pred == 0))
    total = len(y_test)

    print(f"\nTest Set Breakdown (Total: {total} samples):")
    print(f"{'='*60}")
    print(f"✓ True Negative  (TN):  {tn:4d} ({tn/total*100:5.2f}%)")
    print(f"✓ True Positive  (TP):  {tp:4d} ({tp/total*100:5.2f}%)")
    print(f"✗ False Positive (FP):  {fp:4d} ({fp/total*100:5.2f}%)")
    print(f"✗ False Negative (FN):  {fn:4d} ({fn/total*100:5.2f}%)")
    print(f"{'='*60}")
    print(f"Overall Accuracy: {(tn + tp)/total*100:.2f}%")

# ============================================
# STEP 5: RUN ANALYSIS FOR ALL MODELS
# ============================================

print("\n" + "="*60)
print("STARTING RISE ANALYSIS")
print("Generating 10 samples per scenario (TN, TP, FP, FN)")
print("⚠️  Note: RISE computation takes time (2000 masks per image)")
print("="*60)
```

```
# Model 1: Simple CNN
if rise_model1 and 'y_pred1' in dir() and 'y_pred1_prob' in dir():
    analyze_model_rise(
        model_name="Model 1 (Simple CNN)",
        rise_obj=rise_model1,
        y_pred=y_pred1,
        y_pred_prob=y_pred1_prob,
        num_samples=10
    )
else:
    print("\n❌ Skipping Model 1: Missing RISE or predictions")

# Model 2: ResNet
if rise_model2 and 'y_pred2' in dir() and 'y_pred2_prob' in dir():
    analyze_model_rise(
        model_name="Model 2 (ResNet)",
        rise_obj=rise_model2,
        y_pred=y_pred2,
        y_pred_prob=y_pred2_prob,
        num_samples=10
    )
else:
    print("\n❌ Skipping Model 2: Missing RISE or predictions")

# Model 3: VGG16 (if exists)
if rise_model3 and 'y_pred3' in dir() and 'y_pred3_prob' in dir():
    analyze_model_rise(
        model_name="Model 3 (VGG16)",
        rise_obj=rise_model3,
        y_pred=y_pred3,
        y_pred_prob=y_pred3_prob,
        num_samples=10
    )
else:
    print("\n❌ Skipping Model 3: Missing RISE or predictions")

print("\n" + "="*60)
print("✅ RISE ANALYSIS COMPLETE!")
print("="*60)

print("\n💡 KEY IMPROVEMENTS FOR SMOOTH HEATMAPS:")
print("=" * 60)
print("✓ Using LINEAR interpolation for mask upsampling")
print("✓ Gaussian blur (15x15 kernel) to reduce patchiness")
print("✓ Batch processing for efficiency")
print("✓ Proper normalization for consistent visualization")
print("=" * 60)

print("\n📊 INTERPRETATION GUIDE:")
print("=" * 60)
print("• RED/YELLOW regions = High importance for prediction")
print("• BLUE regions       = Low importance")
print("• Overlay shows important regions on original image")
print("• Compare patterns across TN, TP, FP, FN scenarios")
print("=" * 60)
```

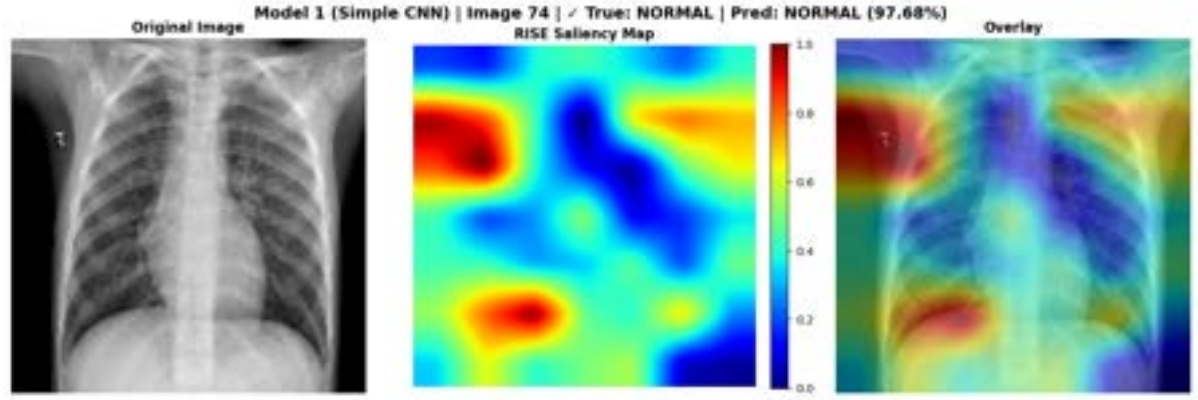
```
============================================================
RISE - EXPLAINABLE AI FOR ALL 3 MODELS
Improved Implementation with Smooth Heatmaps
============================================================

============================================================
INITIALIZING RISE FOR ALL MODELS
============================================================

Model 1 (Simple CNN):
✓ Model 1 (Simple CNN): RISE initialized with 2000 masks

Model 2 (ResNet):
✓ Model 2 (ResNet): RISE initialized with 2000 masks

Model 3 (VGG16):
✓ Model 3 (VGG16): RISE initialized with 2000 masks

============================================================
RISE INITIALIZATION COMPLETE
============================================================

============================================================
STARTING RISE ANALYSIS
Generating 10 samples per scenario (TN, TP, FP, FN)
⚠️  Note: RISE computation takes time (2000 masks per image)
============================================================

============================================================
RISE ANALYSIS: Model 1 (Simple CNN)
============================================================

============================================================
✓ SCENARIO 1: True Negative (Normal → Normal)
============================================================
Found 115 cases, showing 10 examples

  Sample 1/10 - Image 170
```



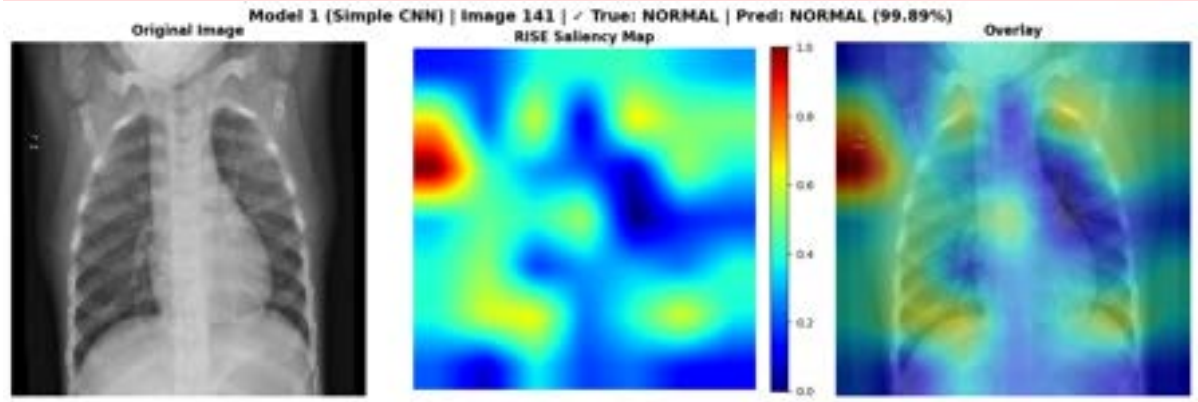

```
  Sample 2/10 - Image 9
```



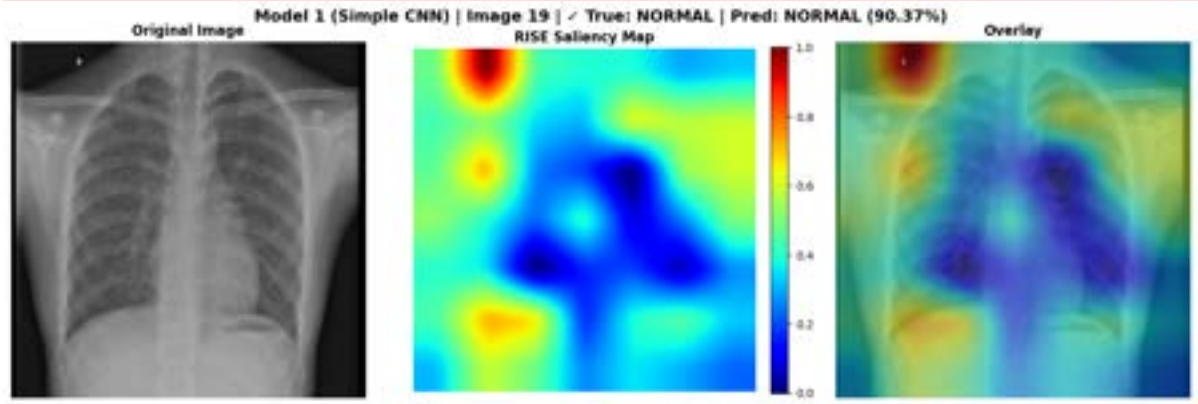
Sample 3/10 — Image 74



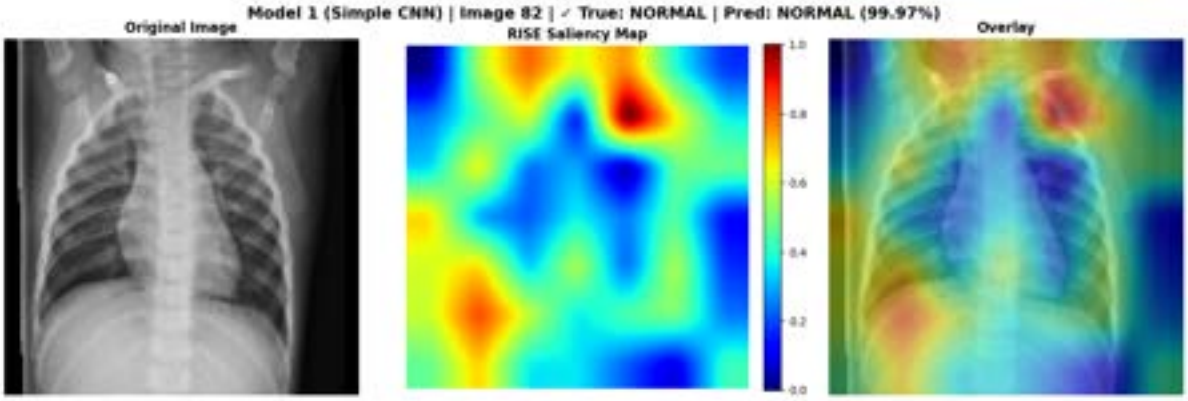
Sample 4/10 — Image 141



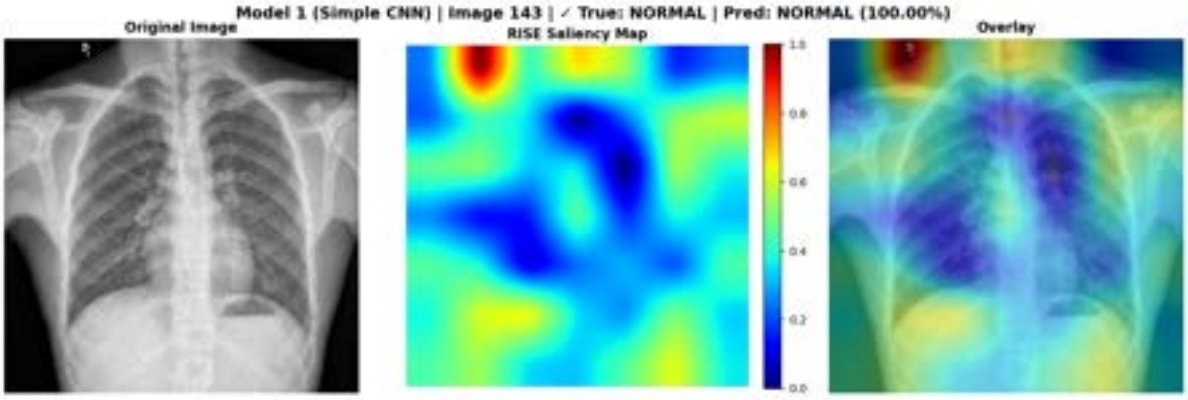
Sample 5/10 — Image 19



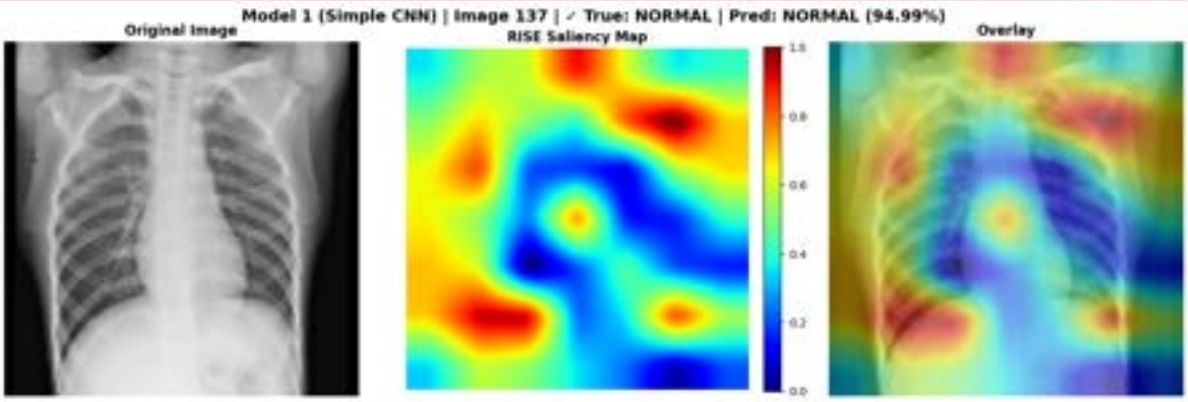
Sample 6/10 — Image 82



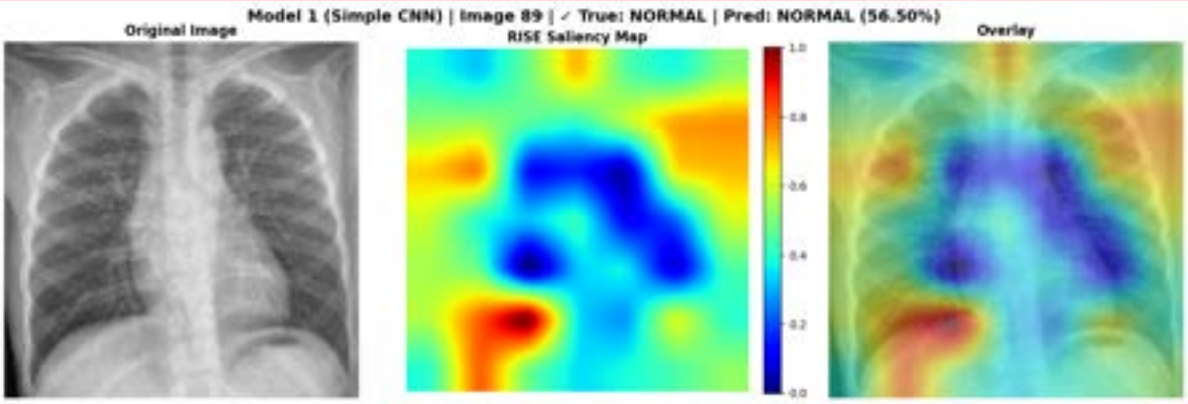
Sample 7/10 — Image 143



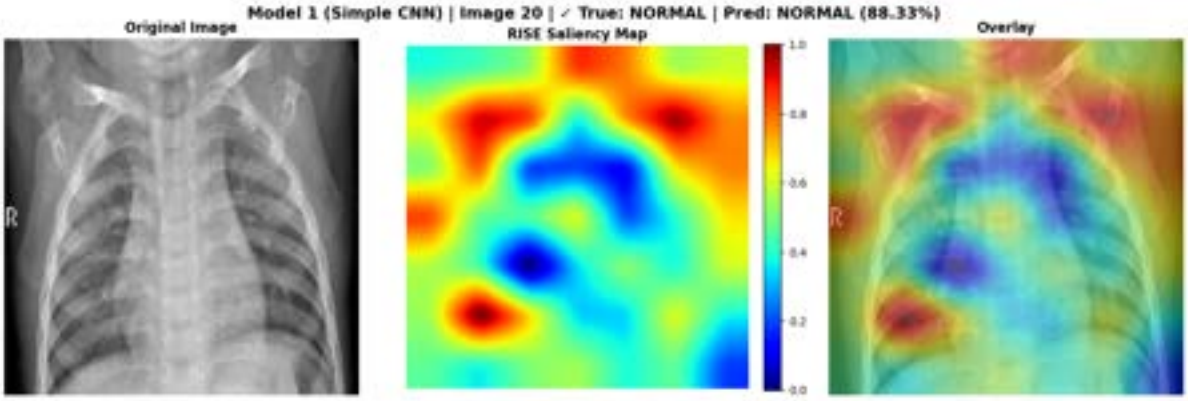
Sample 8/10 — Image 137



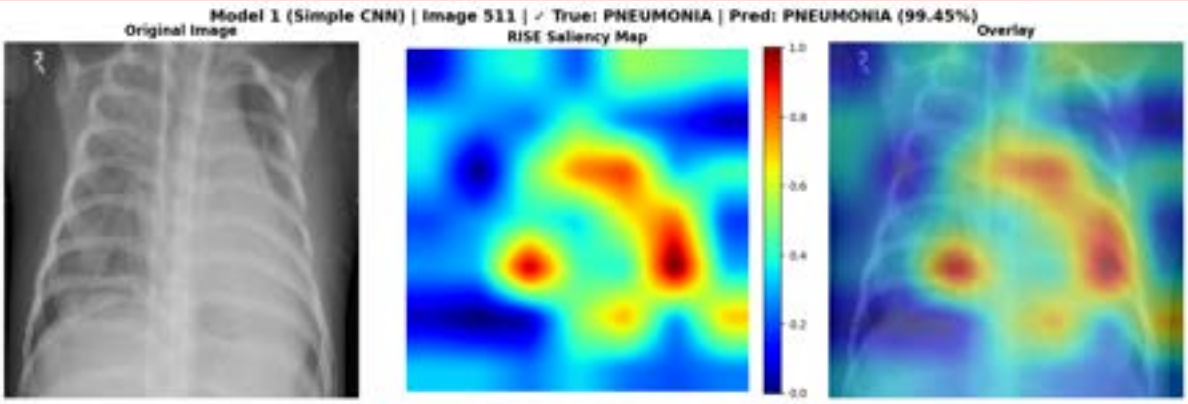
Sample 9/10 — Image 89



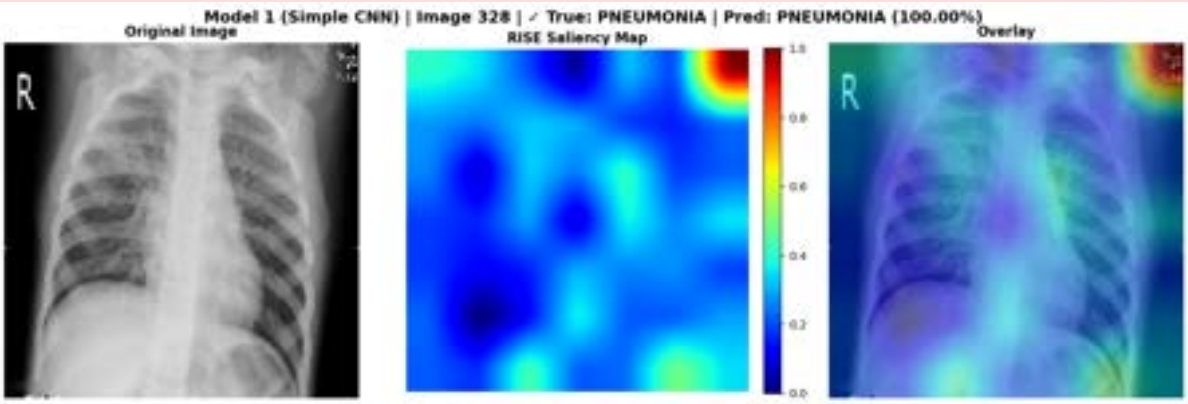
Sample 10/10 — Image 20



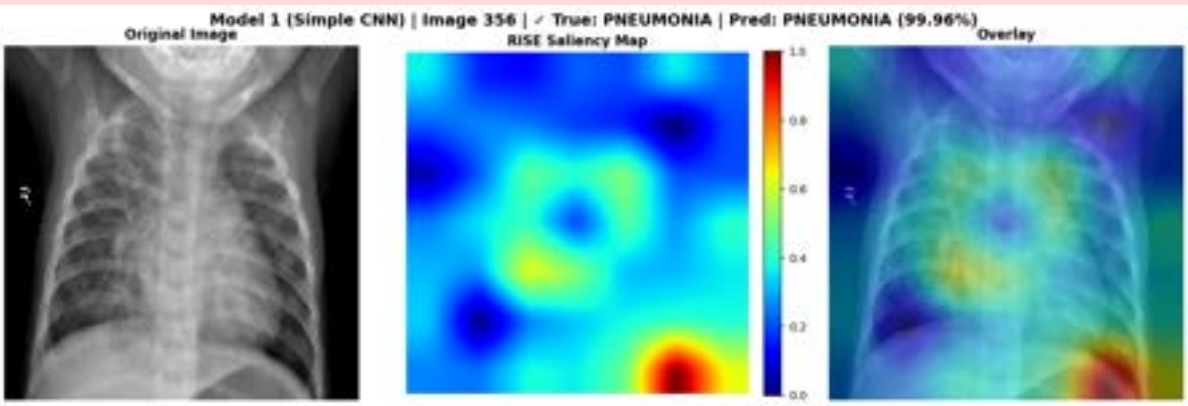

```
============================================================
✓ SCENARIO 2: True Positive (Pneumonia → Pneumonia)
============================================================
Found 387 cases, showing 10 examples

  Sample 1/10 – Image 511
```



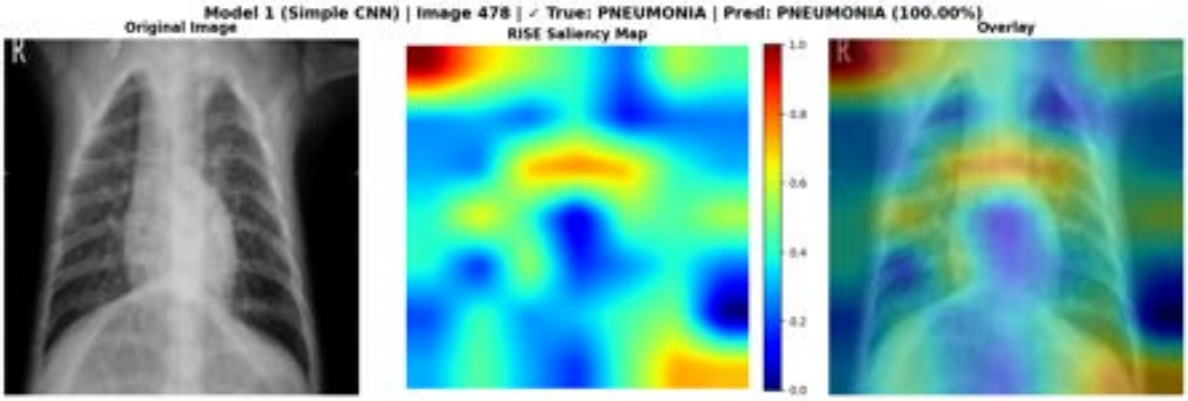

```
  Sample 2/10 – Image 328
```



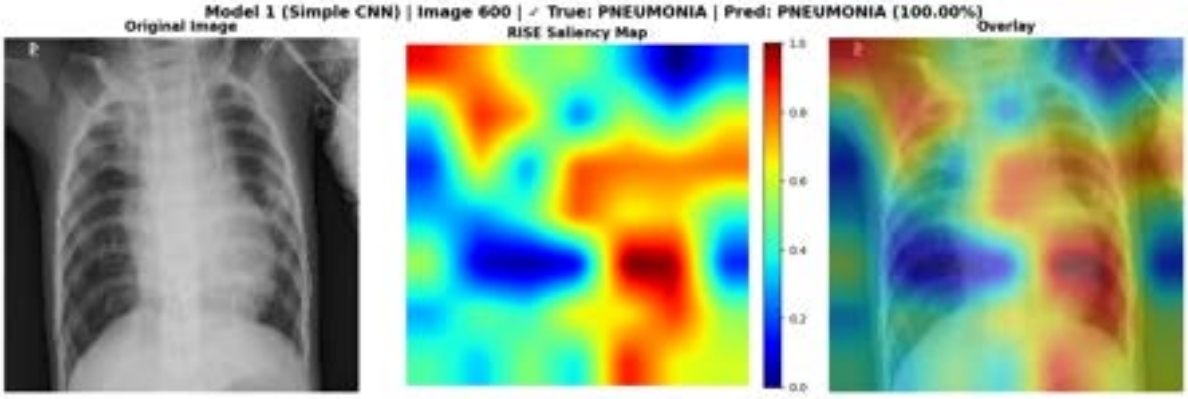

```
  Sample 3/10 – Image 356
```



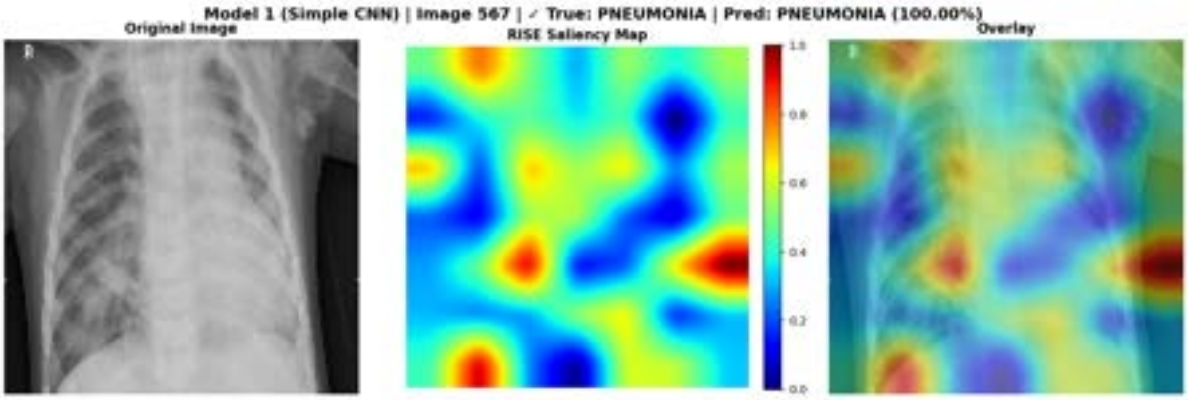

```
  Sample 4/10 – Image 478
```



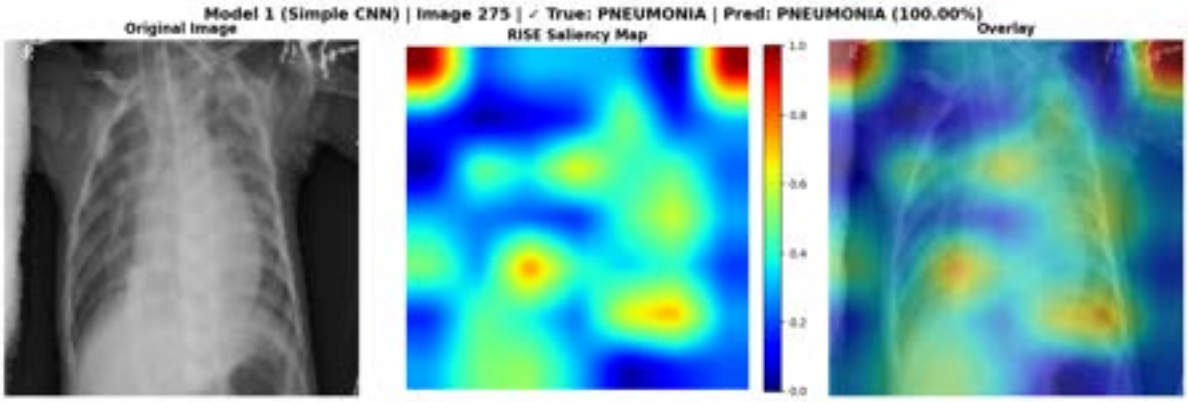
Sample 5/10 — Image 600



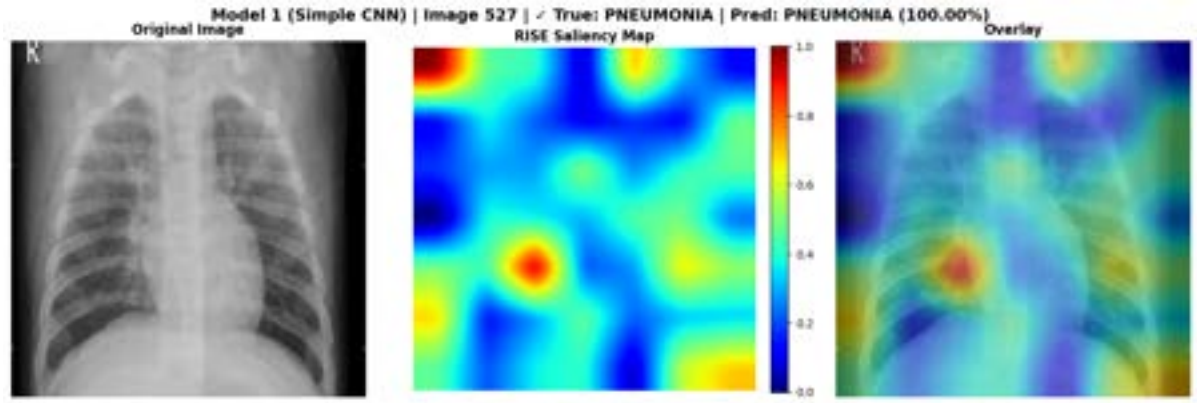
Sample 6/10 — Image 567



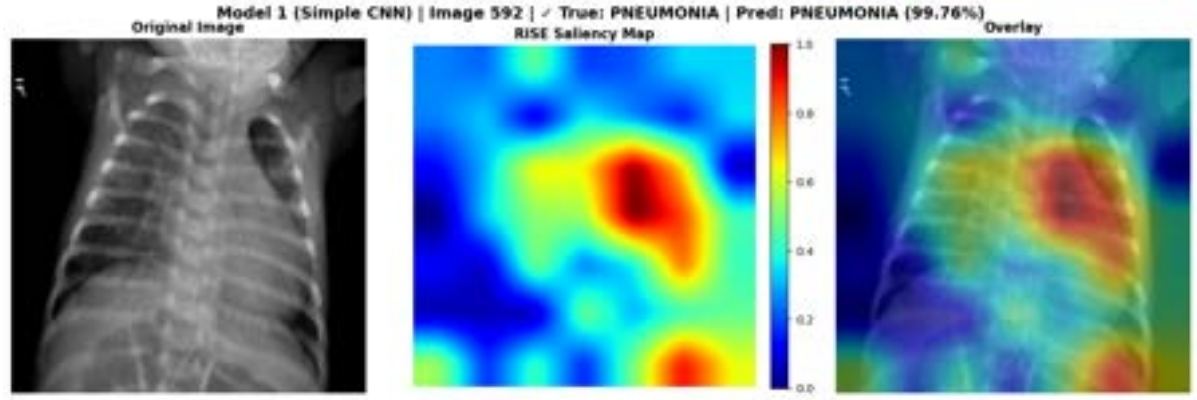
Sample 7/10 — Image 275



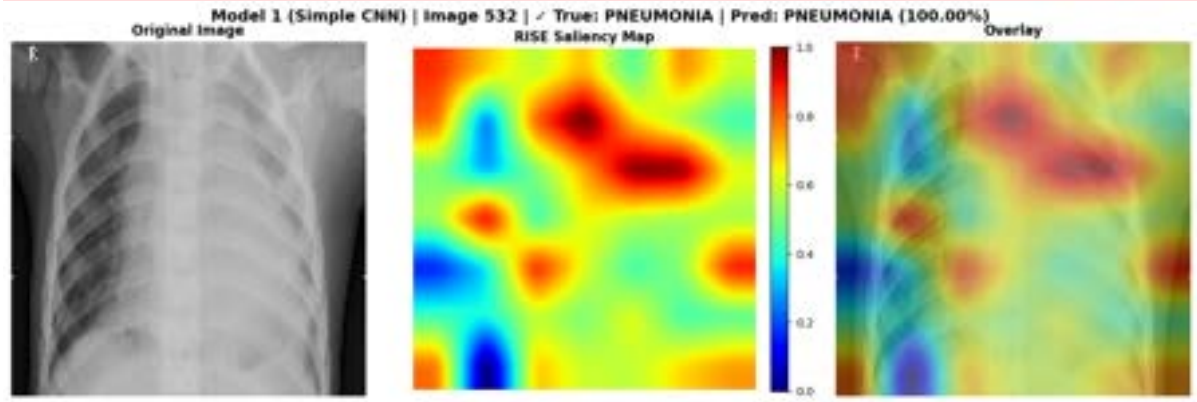
Sample 8/10 — Image 527



Sample 9/10 — Image 592

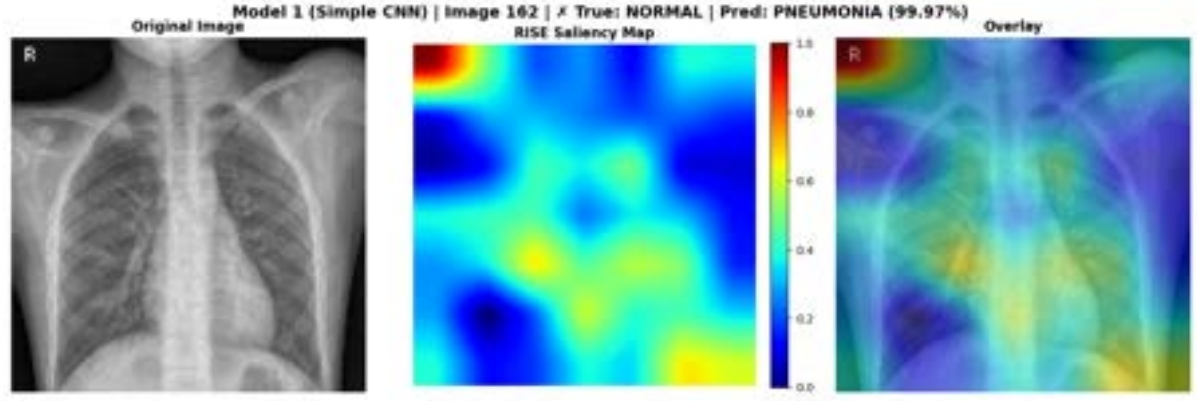


Sample 10/10 — Image 532

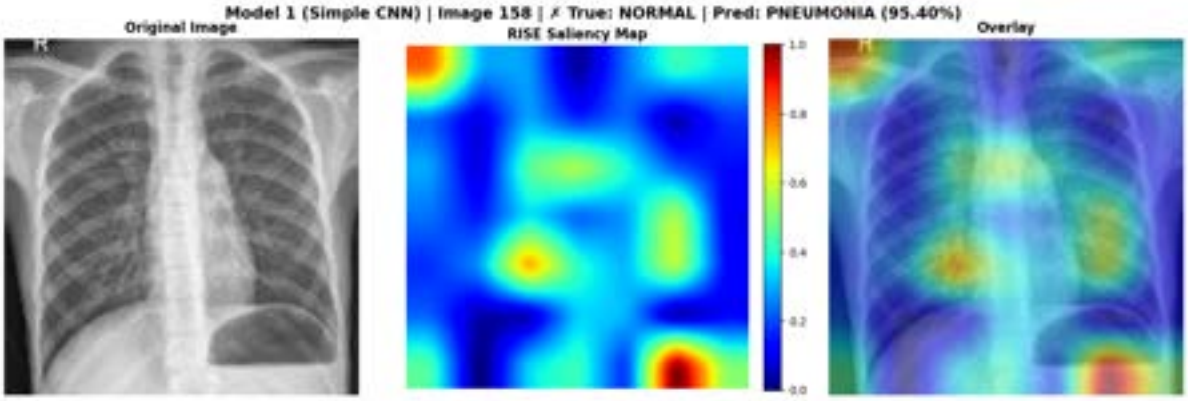



```
============================================================
✗ SCENARIO 3: False Positive (Normal → Pneumonia)
============================================================
Found 119 cases, showing 10 examples
```

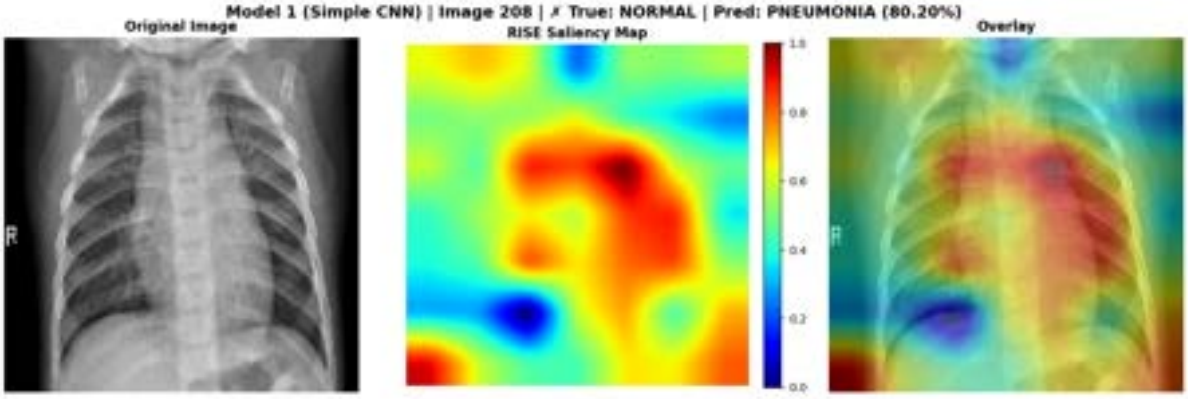
Sample 1/10 — Image 162



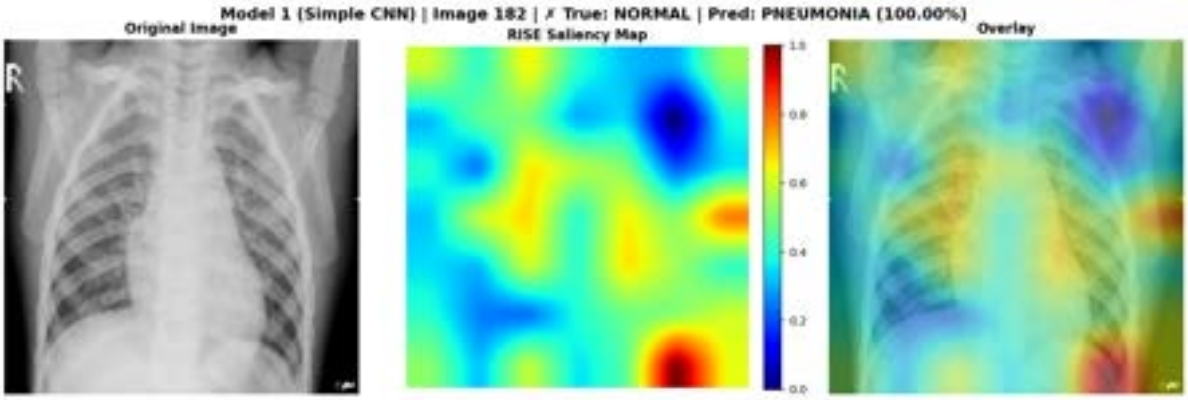
Sample 2/10 — Image 158



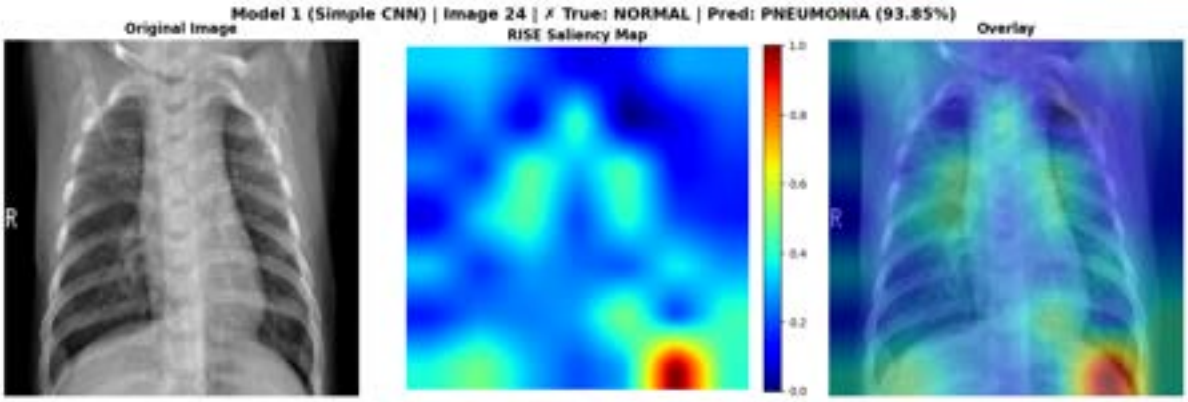
Sample 3/10 – Image 208



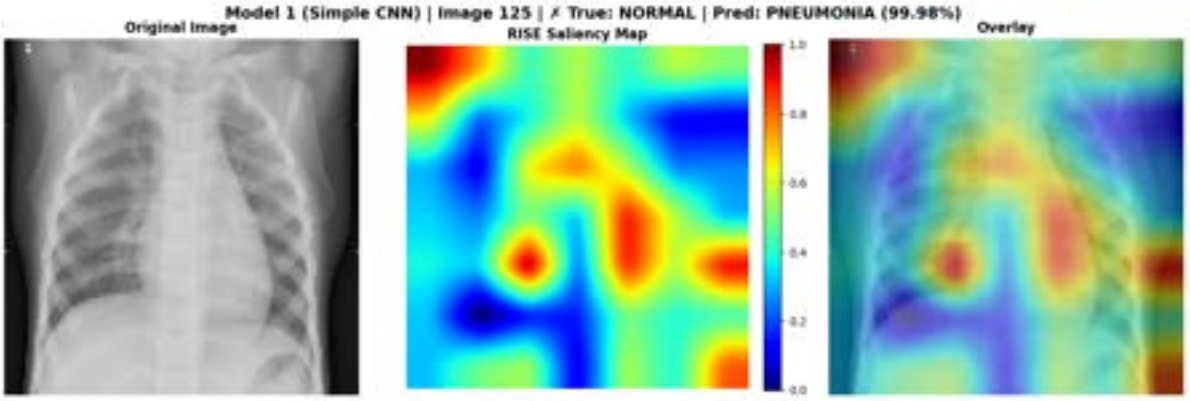
Sample 4/10 – Image 182



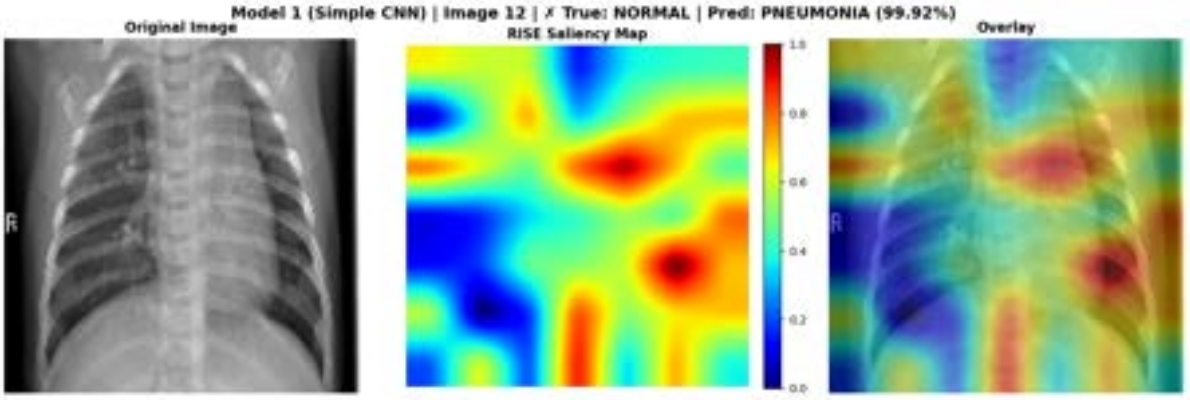
Sample 5/10 – Image 24



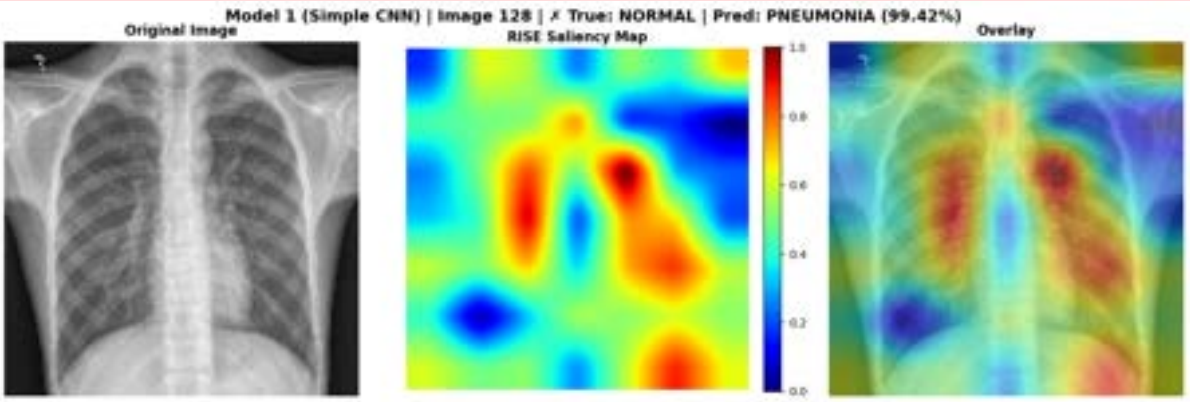
Sample 6/10 – Image 125



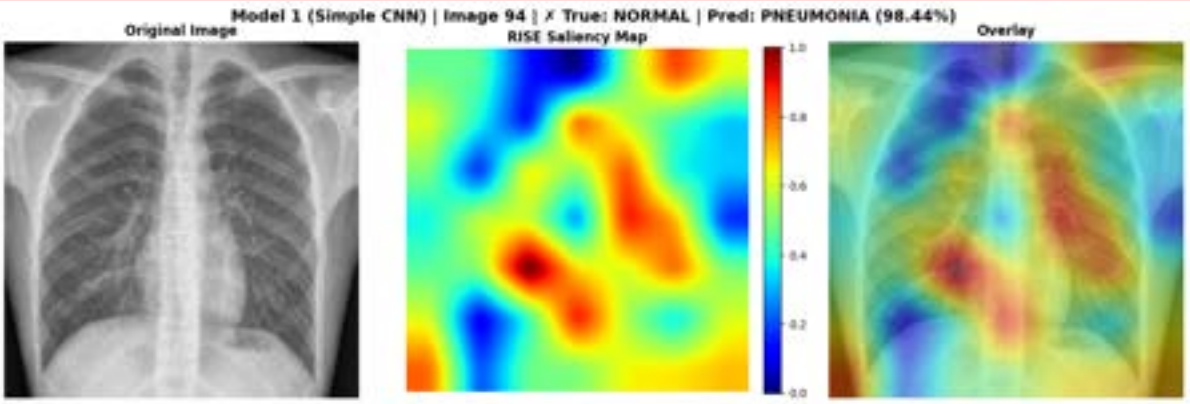
Sample 7/10 — Image 12



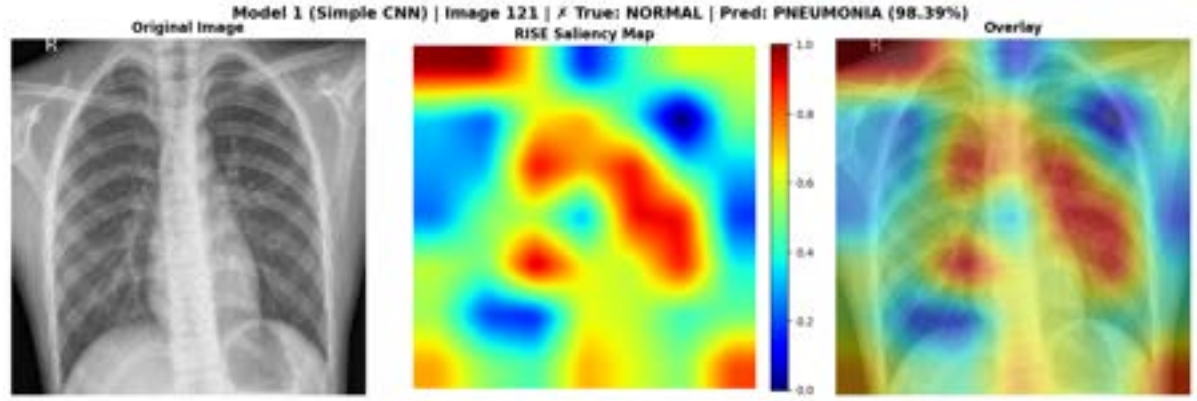
Sample 8/10 — Image 128



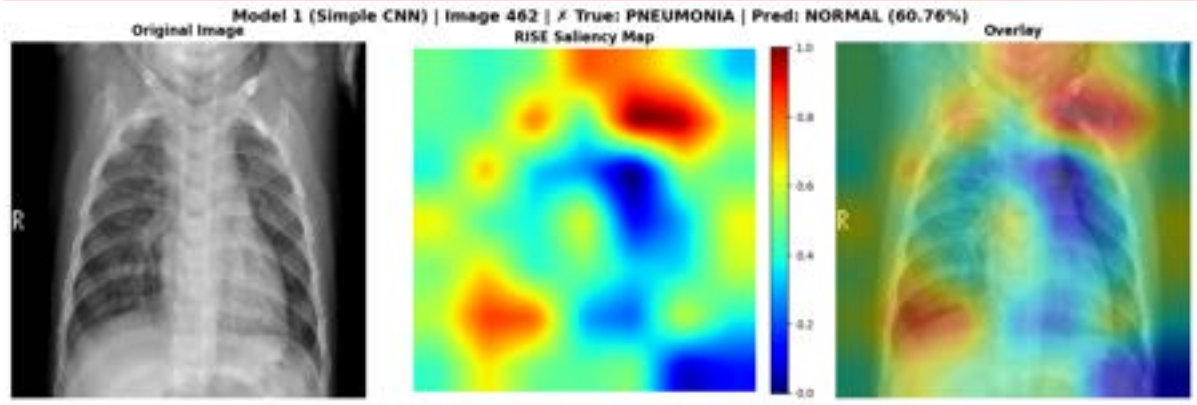
Sample 9/10 — Image 94



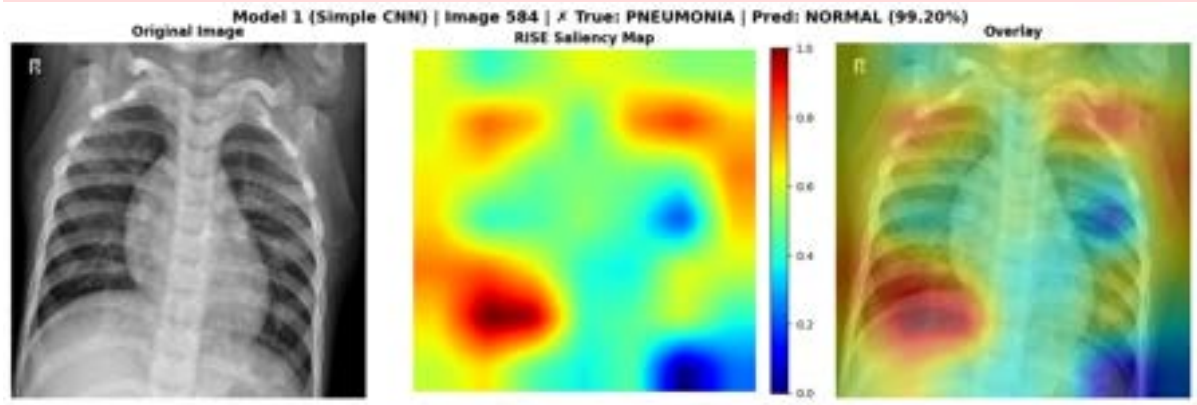
Sample 10/10 — Image 121



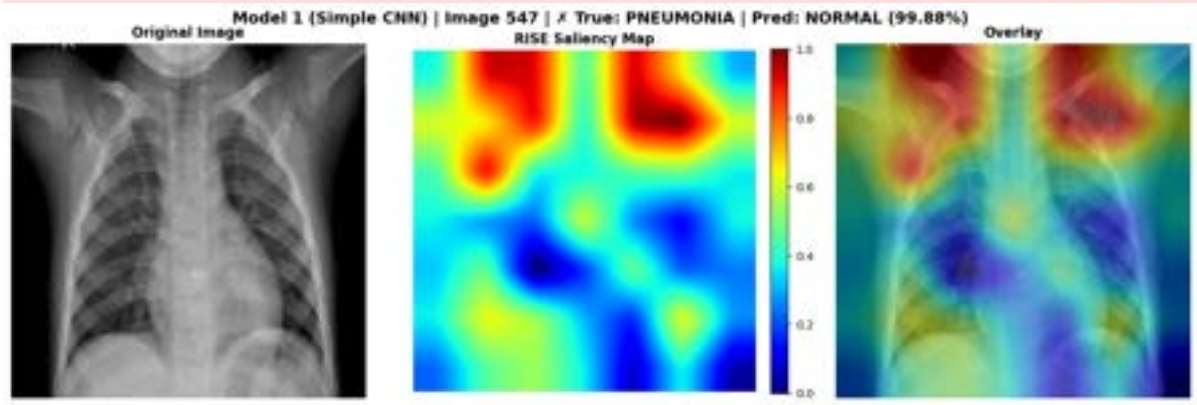

```
============================================================
✗ SCENARIO 4: False Negative (Pneumonia → Normal)
============================================================
Found 3 cases, showing 3 examples

  Sample 1/3 – Image 462
```



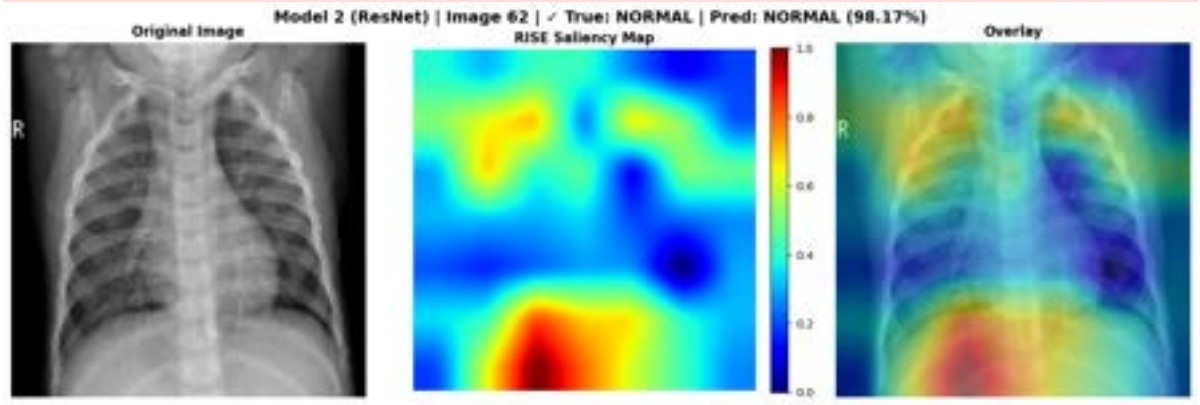

```
  Sample 2/3 – Image 584
```



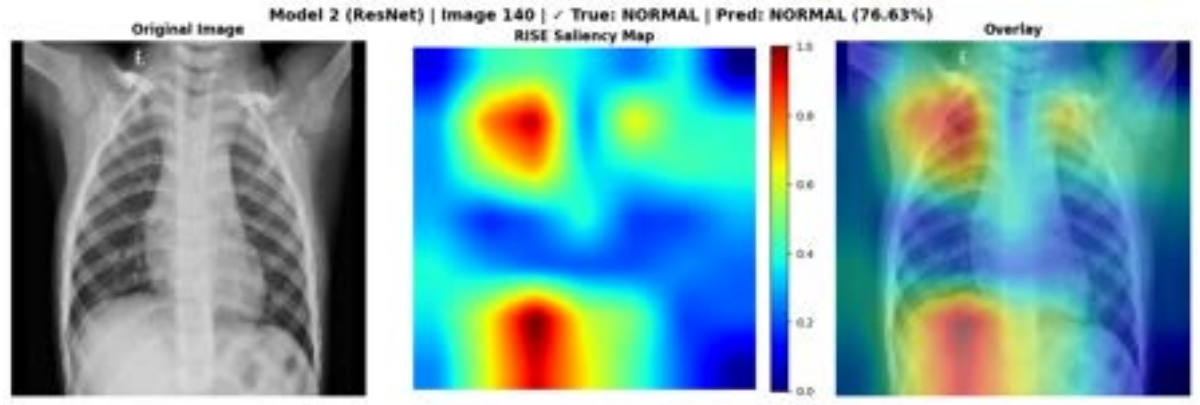

```
  Sample 3/3 – Image 547
```



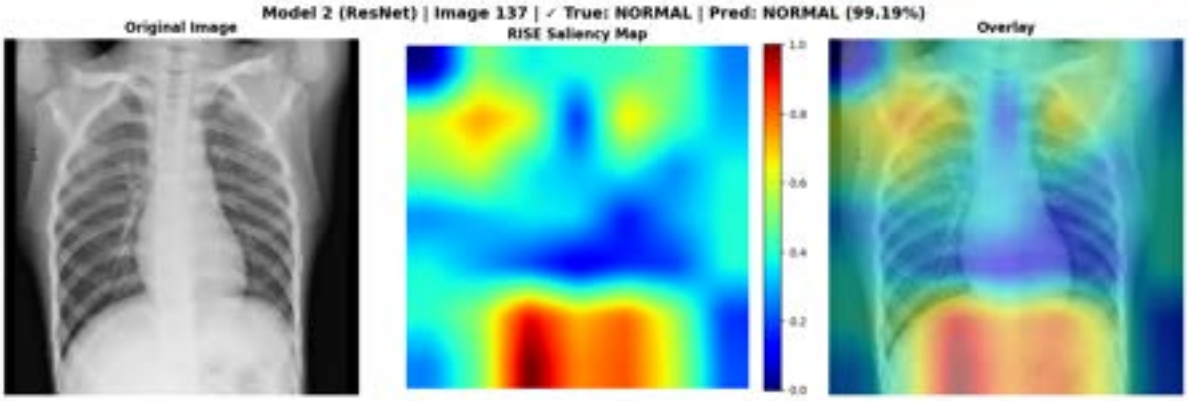
```
============================================================
Model 1 (Simple CNN) - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):   115 (18.43%)
✓ True Positive  (TP):   387 (62.02%)
✗ False Positive (FP):   119 (19.07%)
✗ False Negative (FN):     3 ( 0.48%)
============================================================
Overall Accuracy: 80.45%

============================================================
RISE ANALYSIS: Model 2 (ResNet)
============================================================

============================================================
✓ SCENARIO 1: True Negative (Normal → Normal)
============================================================
Found 105 cases, showing 10 examples

  Sample 1/10 - Image 62
```



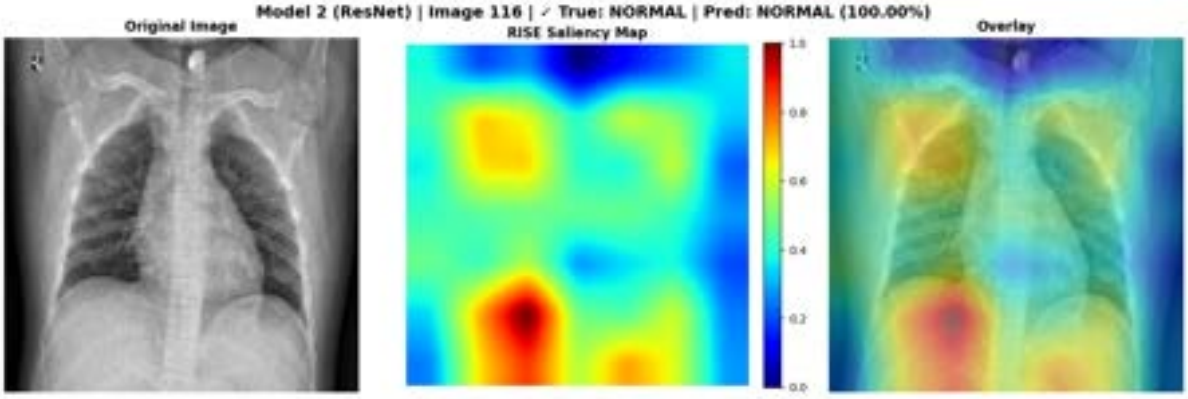

```
  Sample 2/10 - Image 140
```



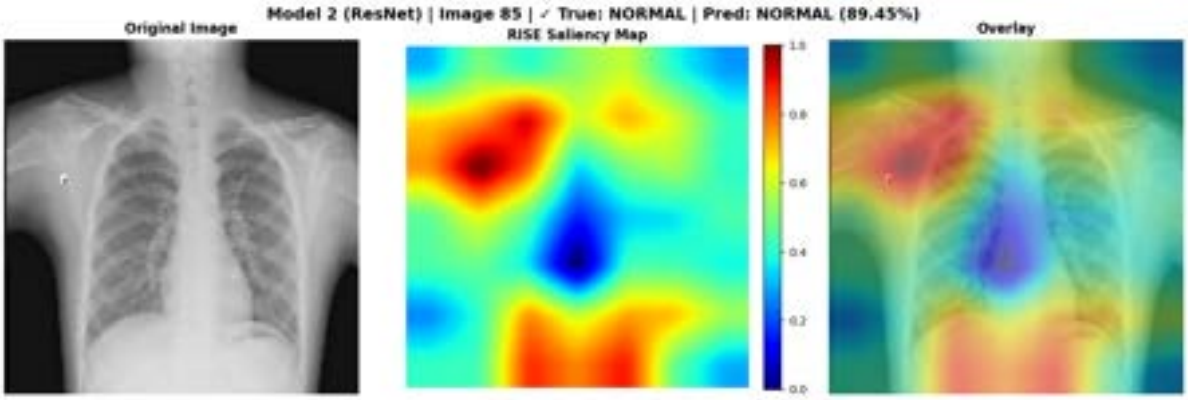

```
  Sample 3/10 - Image 137
```



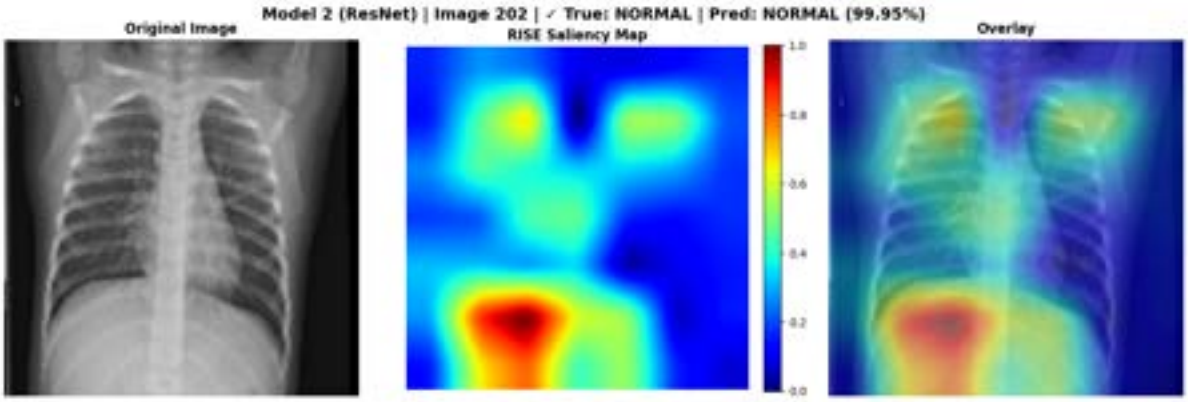
Sample 4/10 – Image 116



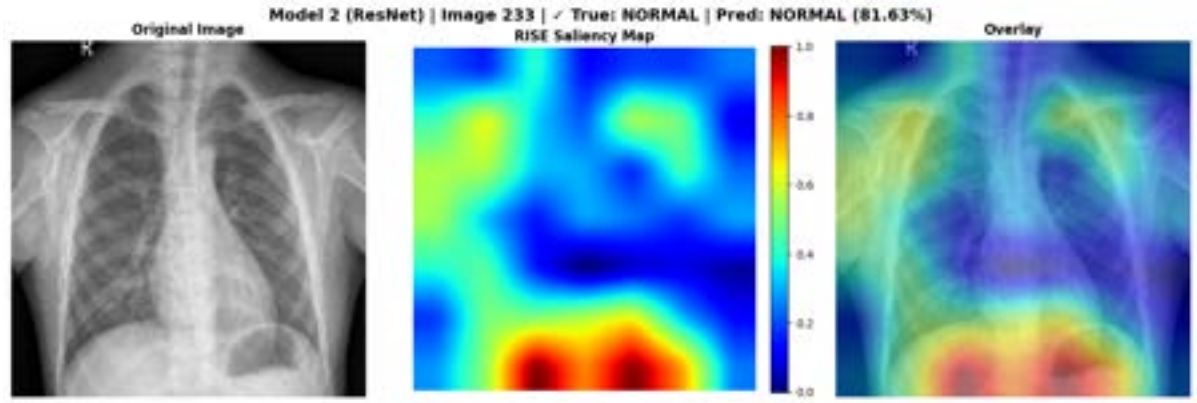
Sample 5/10 – Image 85



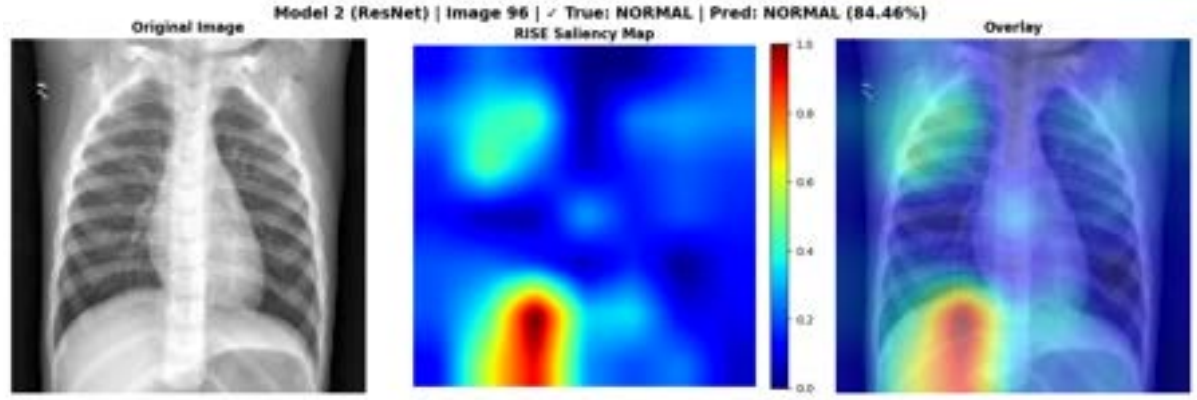
Sample 6/10 – Image 202



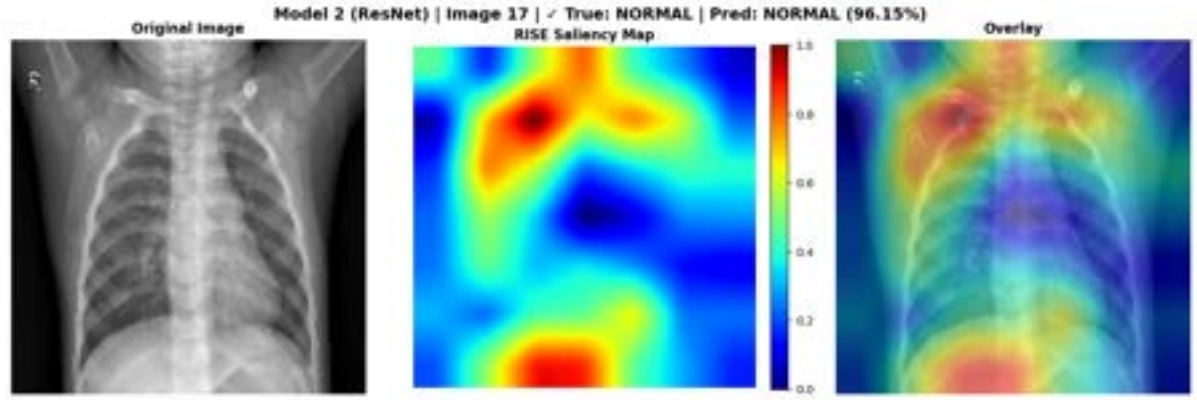
Sample 7/10 – Image 233



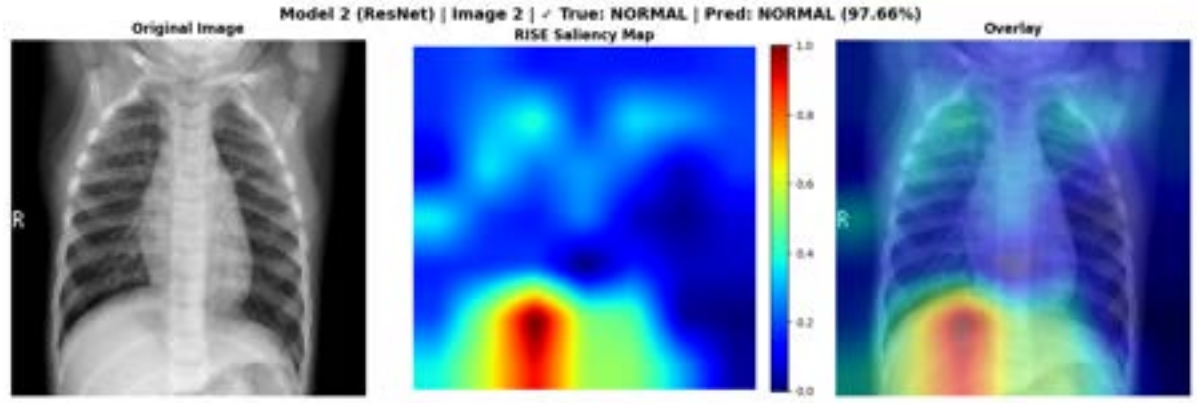
Sample 8/10 — Image 96



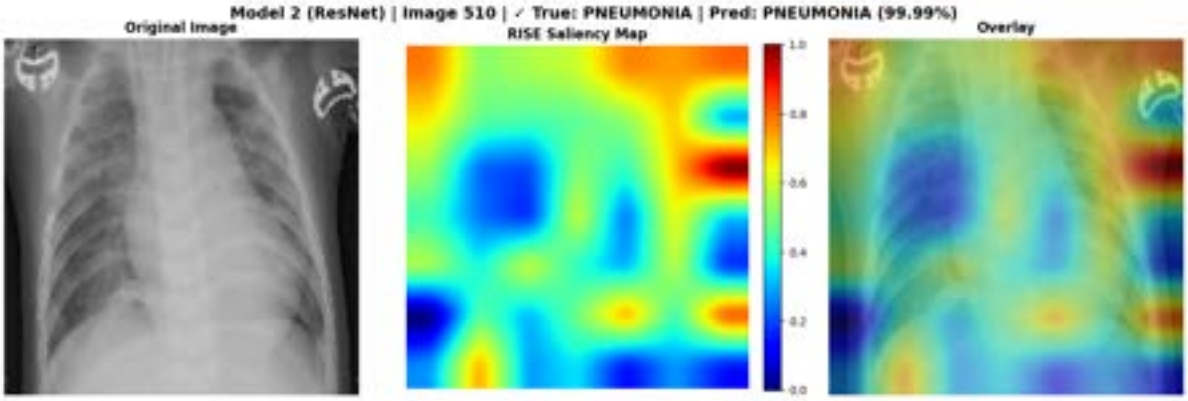
Sample 9/10 — Image 17



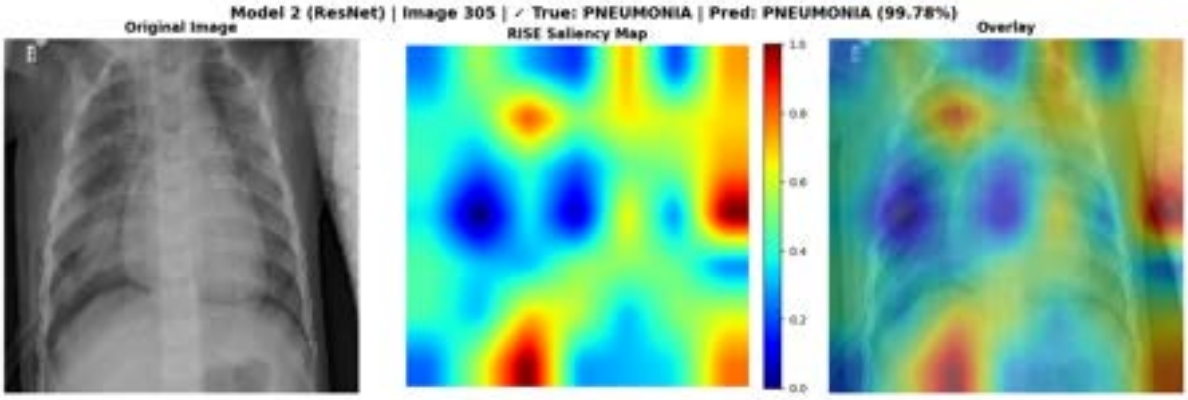
Sample 10/10 — Image 2



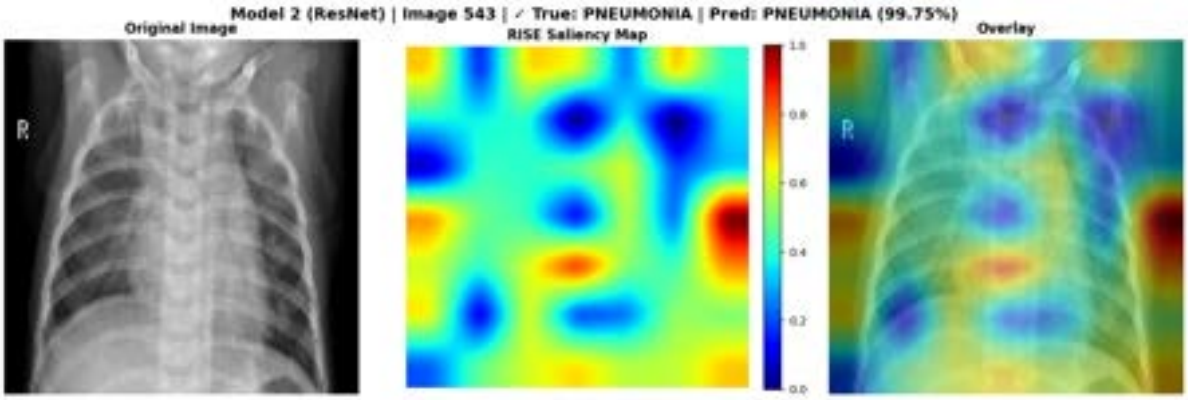

```
============================================================
✓ SCENARIO 2: True Positive (Pneumonia → Pneumonia)
============================================================
Found 386 cases, showing 10 examples

  Sample 1/10 — Image 510
```



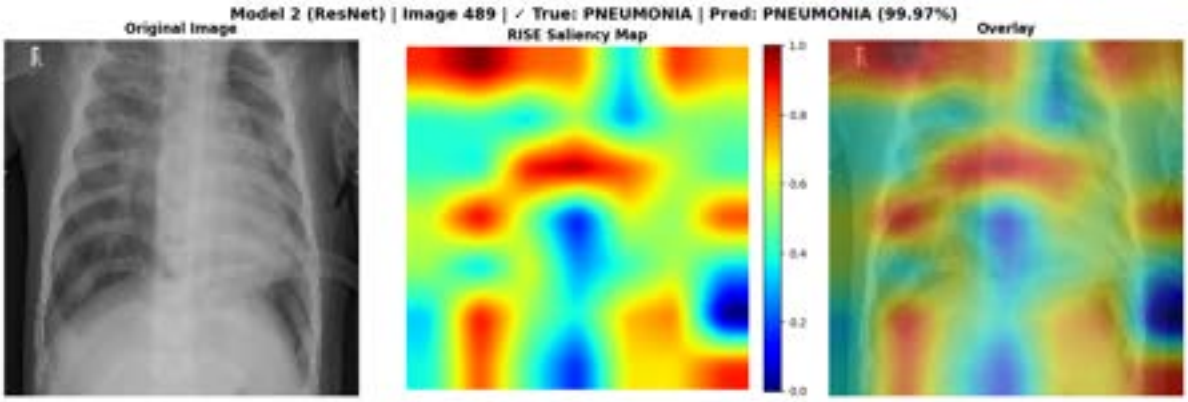
Sample 2/10 — Image 305



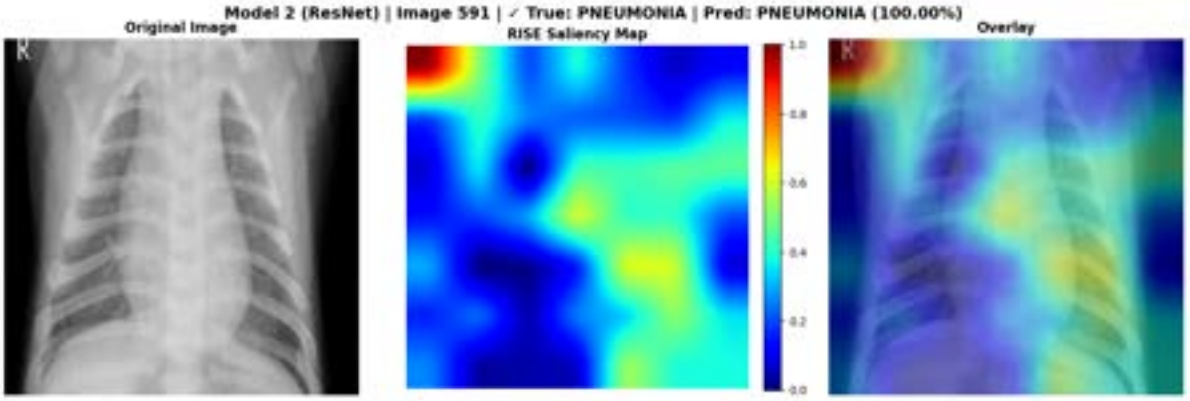
Sample 3/10 — Image 543



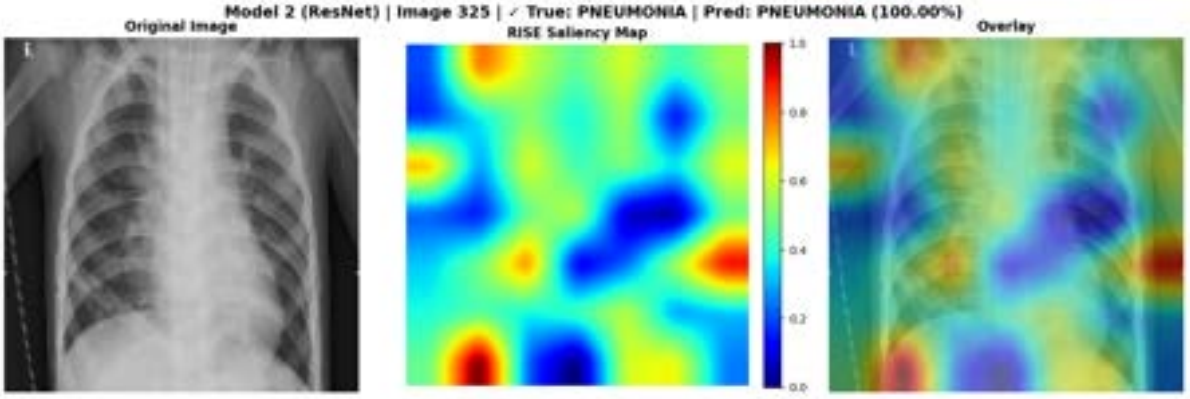
Sample 4/10 — Image 489



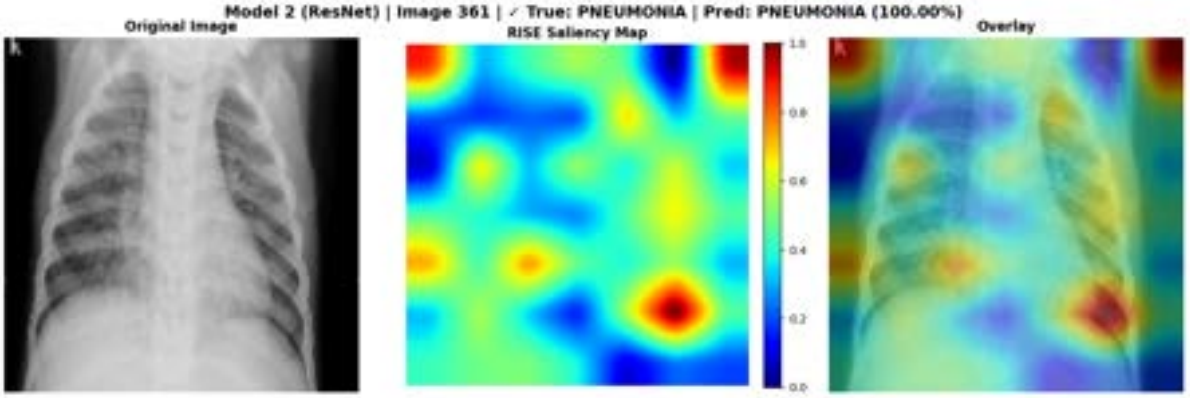
Sample 5/10 — Image 591



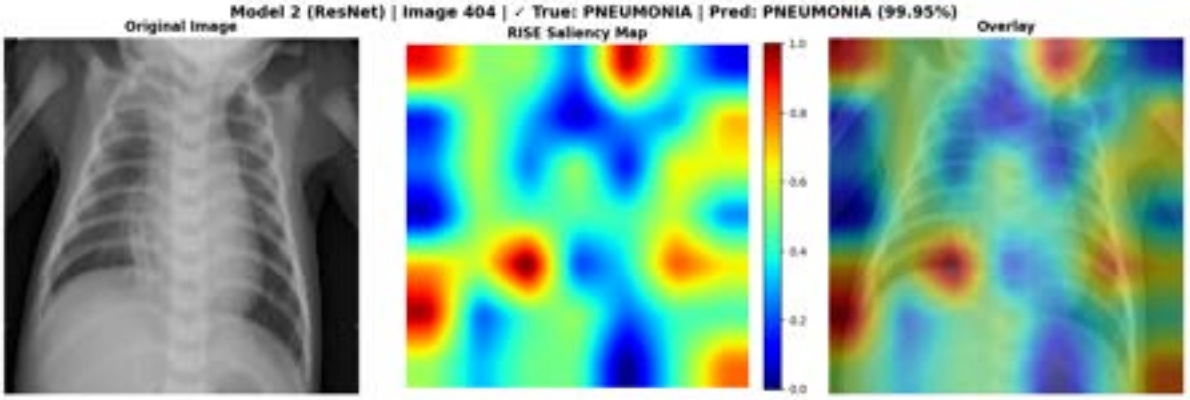
Sample 6/10 — Image 325



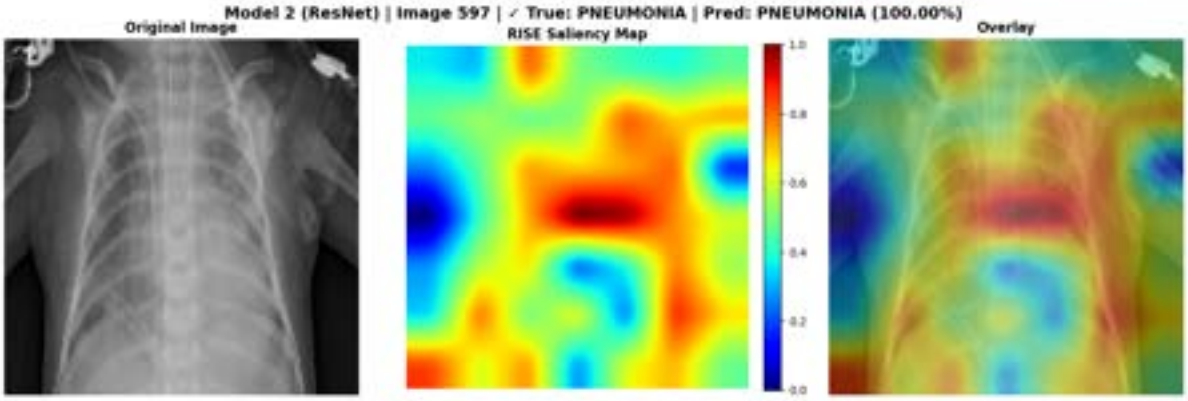
Sample 7/10 — Image 361



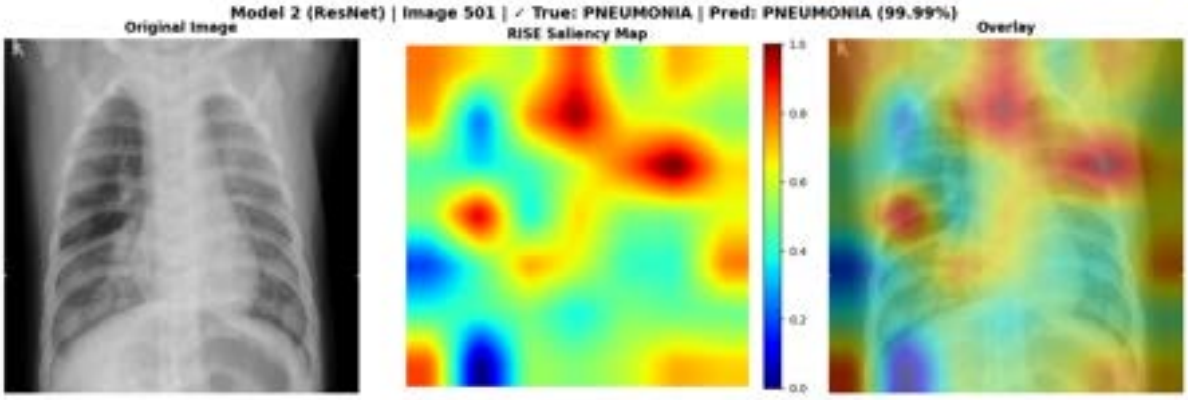
Sample 8/10 — Image 404



Sample 9/10 — Image 597

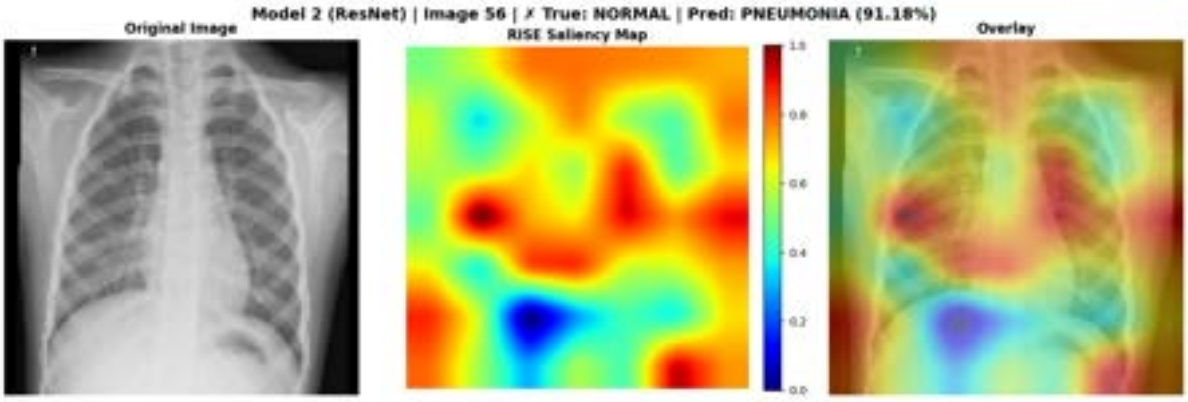


Sample 10/10 – Image 501

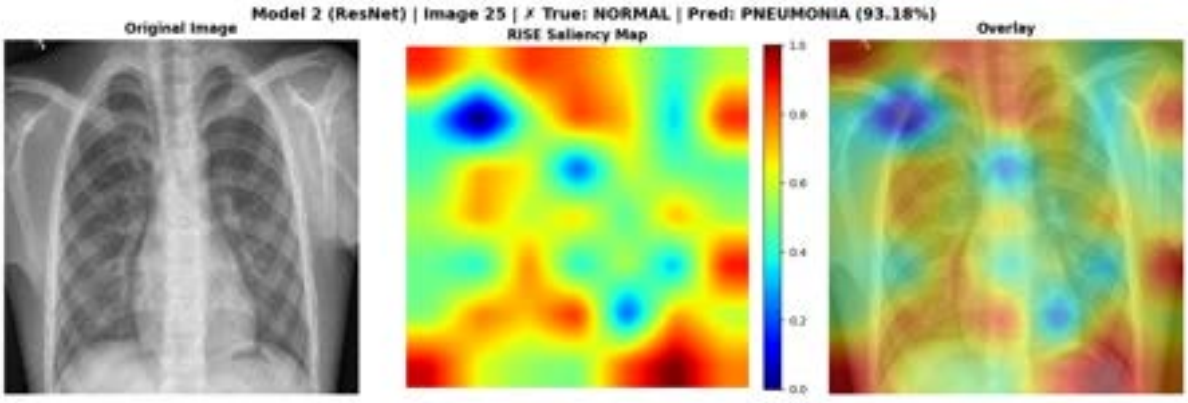



```
============================================================
✗ SCENARIO 3: False Positive (Normal → Pneumonia)
============================================================
Found 129 cases, showing 10 examples
```

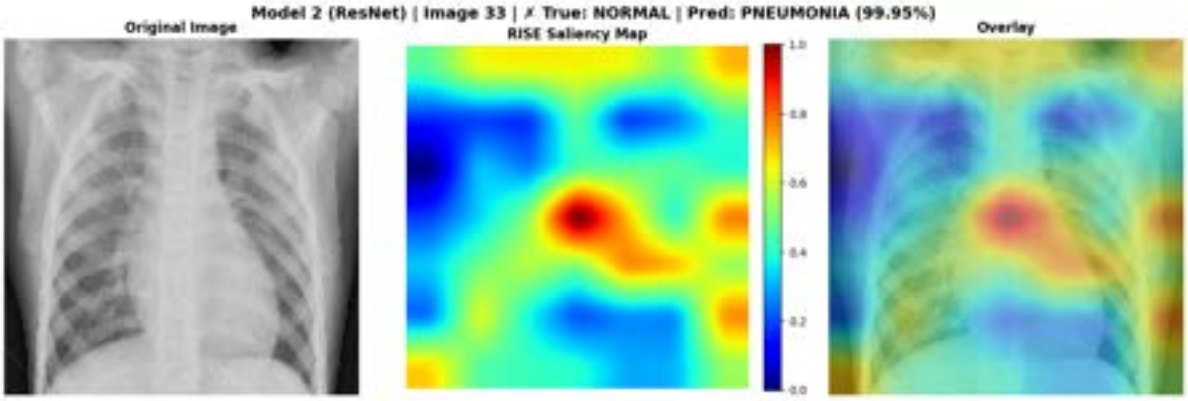
Sample 1/10 – Image 56



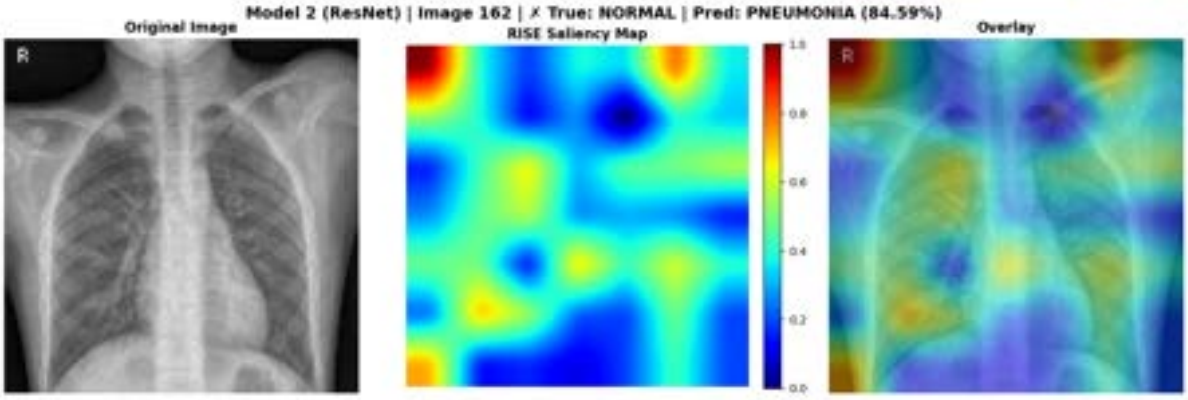
Sample 2/10 – Image 25



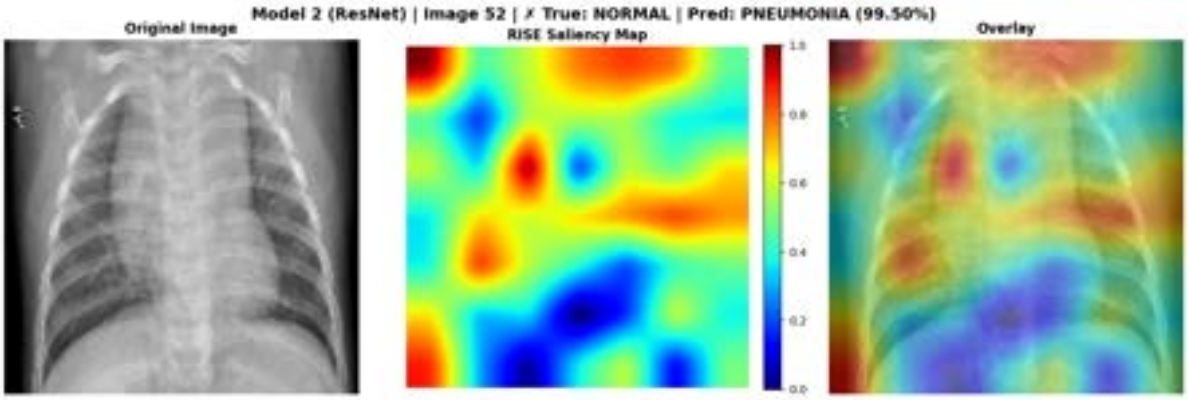
Sample 3/10 – Image 33



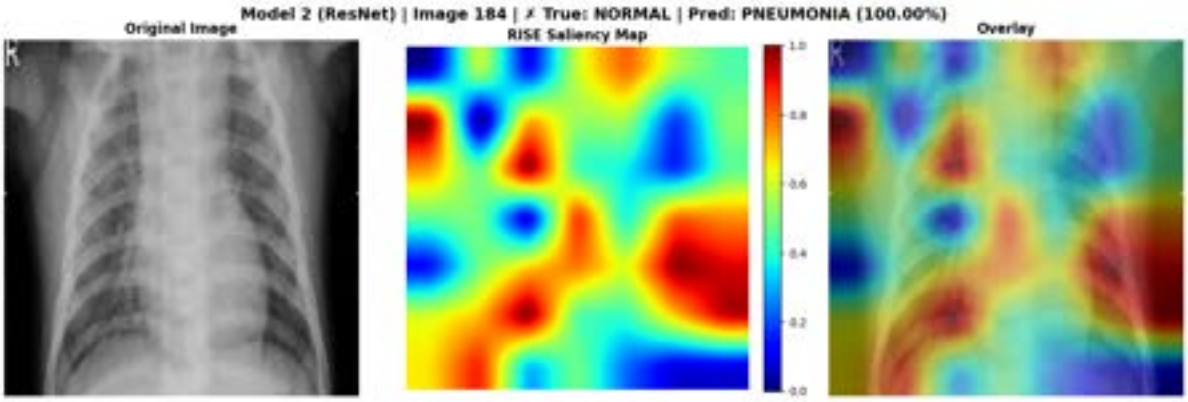
Sample 4/10 — Image 162



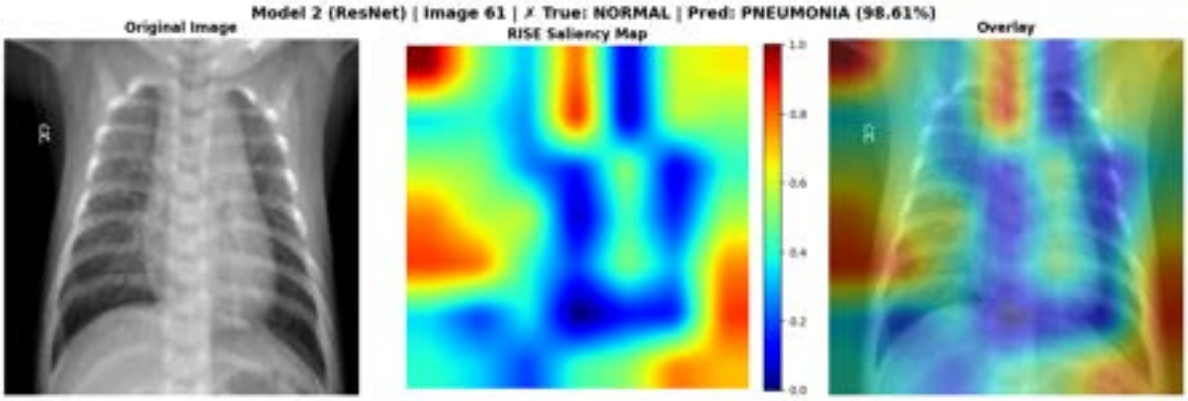
Sample 5/10 — Image 52



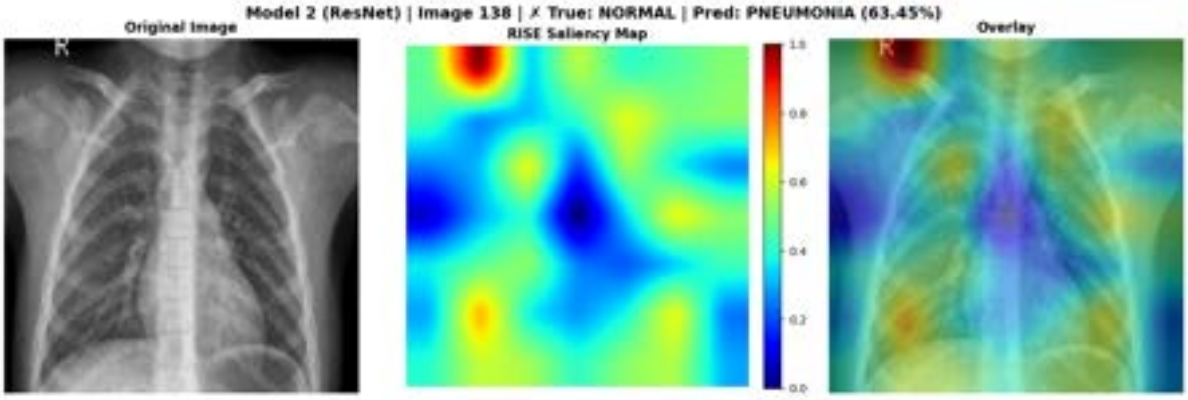
Sample 6/10 — Image 184



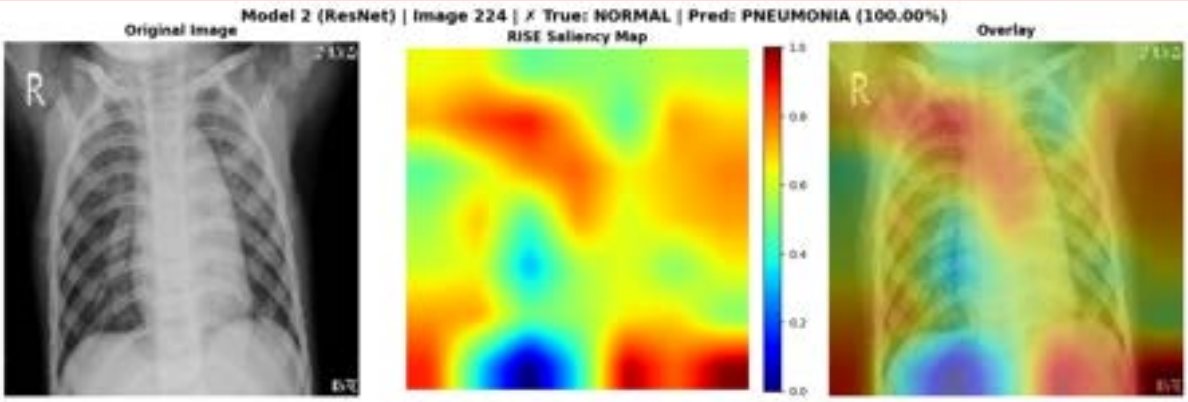
Sample 7/10 — Image 61



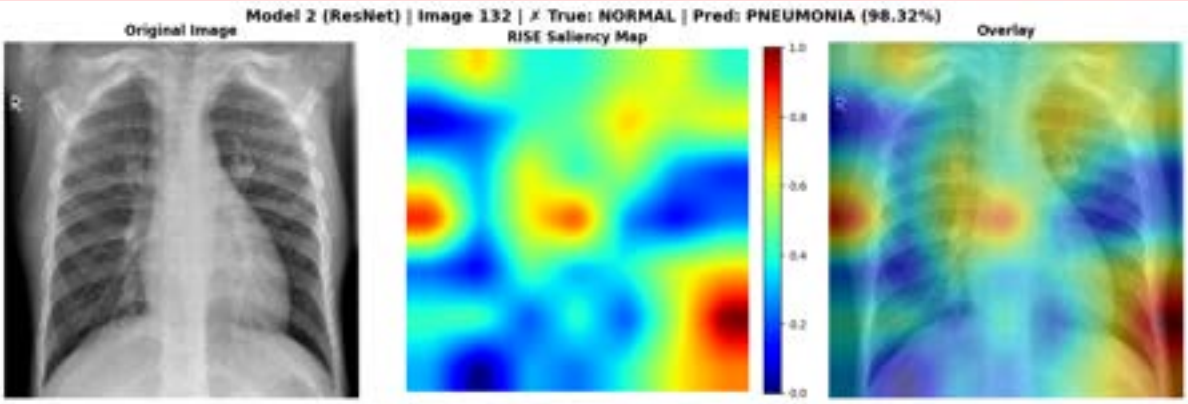
Sample 8/10 — Image 138



Sample 9/10 — Image 224

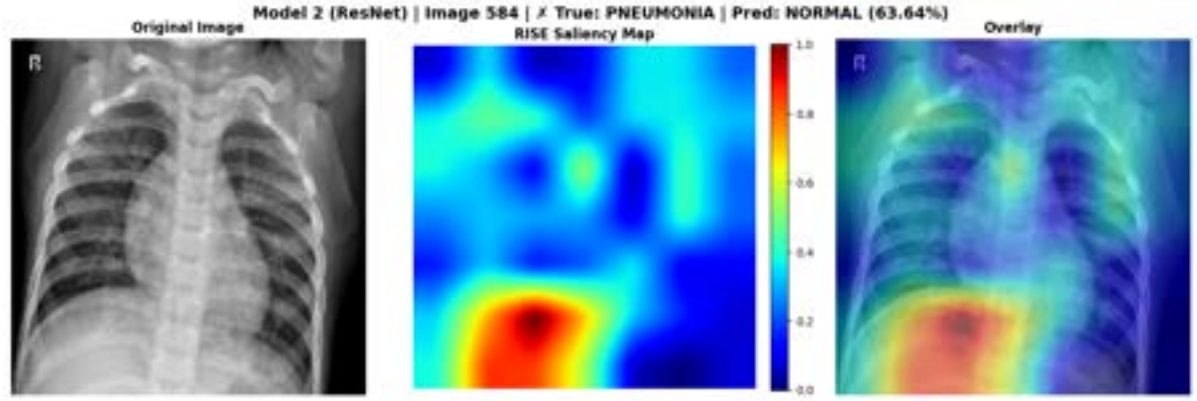


Sample 10/10 — Image 132

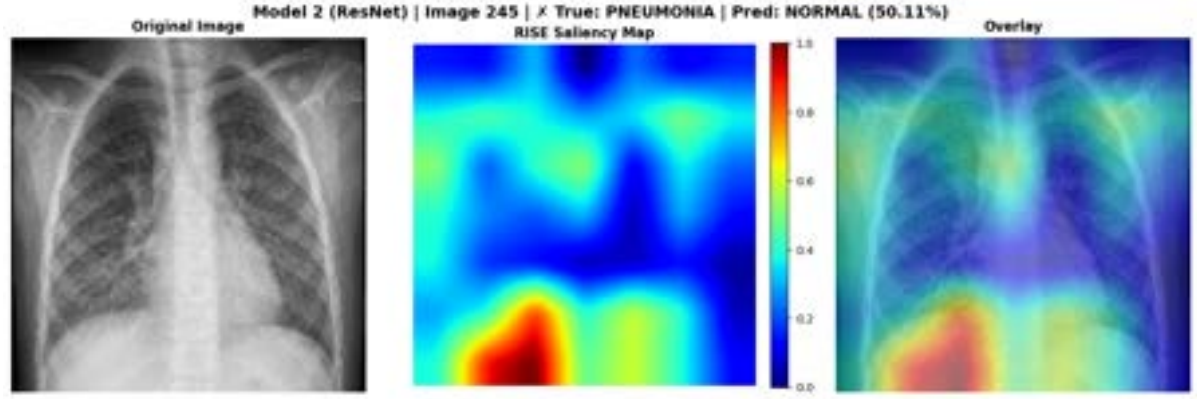



```
============================================================
✗ SCENARIO 4: False Negative (Pneumonia → Normal)
============================================================
Found 4 cases, showing 4 examples
```

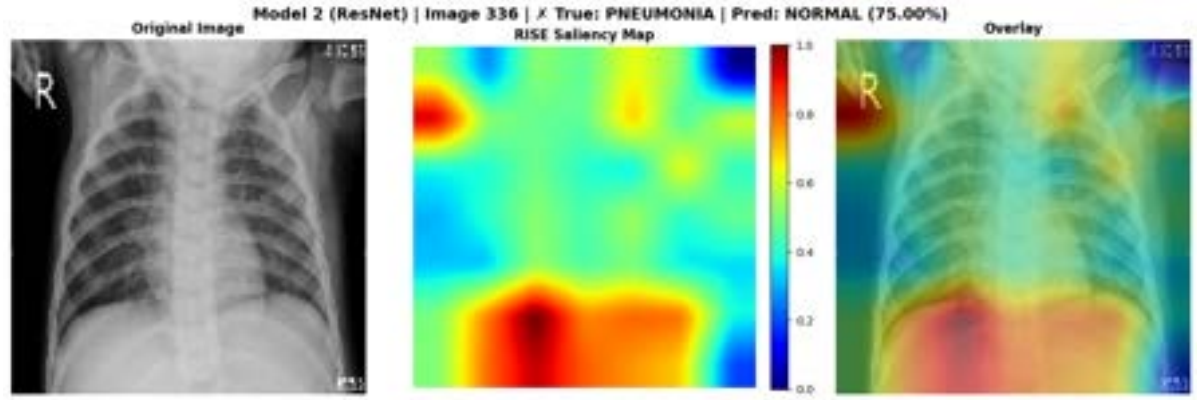
Sample 1/4 — Image 584



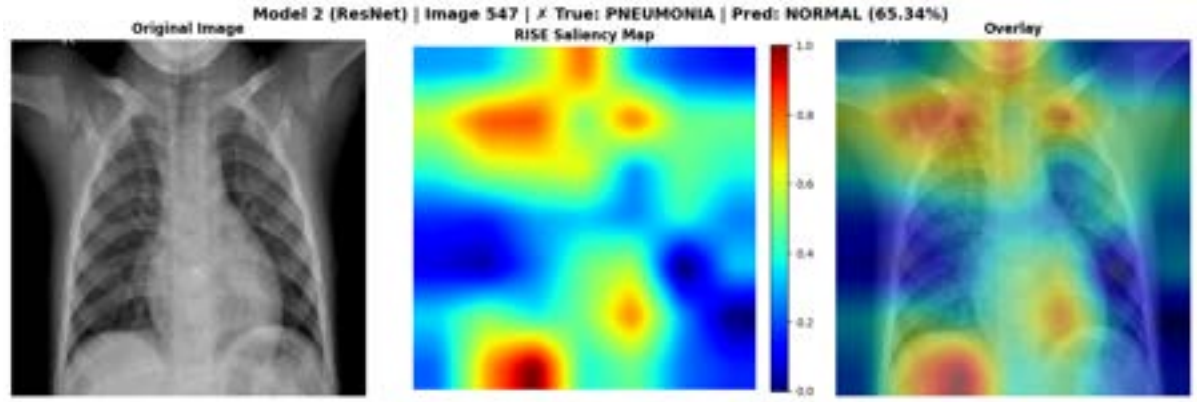
Sample 2/4 — Image 245



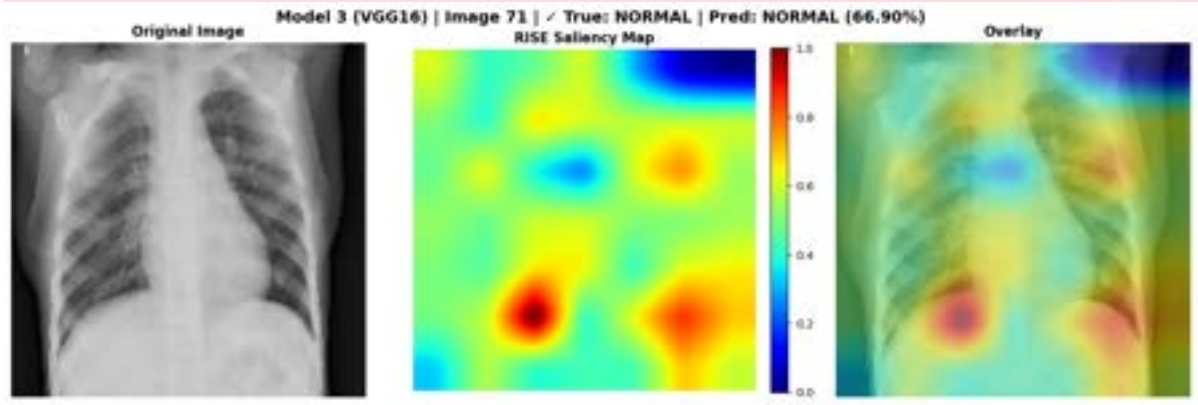
Sample 3/4 — Image 336



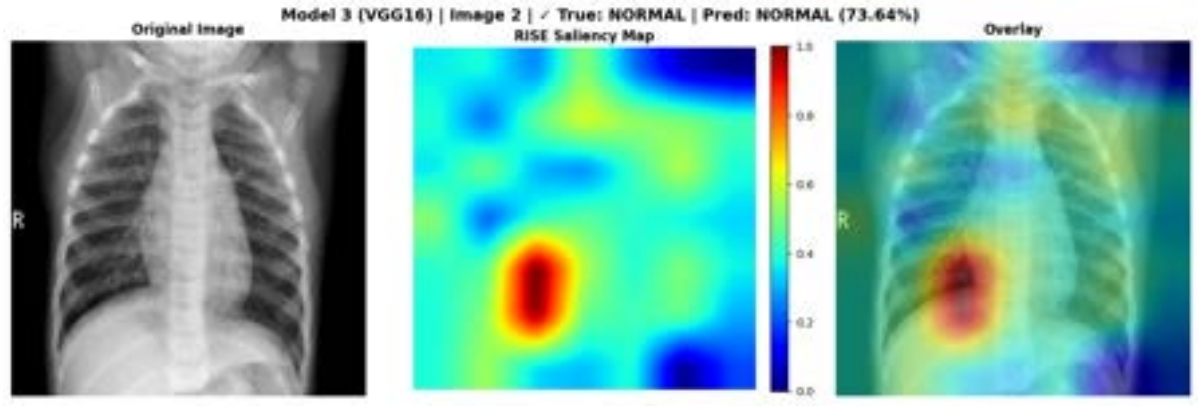
Sample 4/4 — Image 547



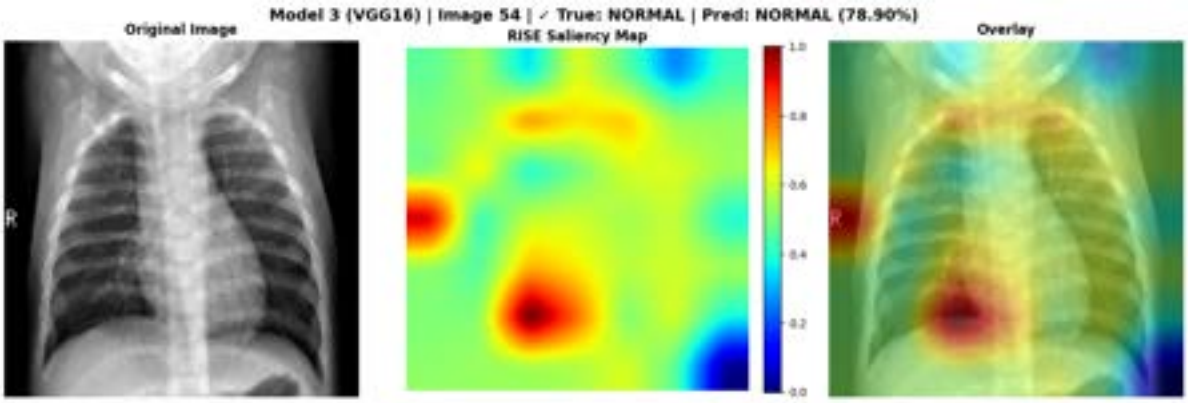
```
============================================================
Model 2 (ResNet) - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):   105 (16.83%)
✓ True Positive  (TP):   386 (61.86%)
✗ False Positive (FP):   129 (20.67%)
✗ False Negative (FN):     4 ( 0.64%)
============================================================
Overall Accuracy: 78.69%

============================================================
RISE ANALYSIS: Model 3 (VGG16)
============================================================

============================================================
✓ SCENARIO 1: True Negative (Normal → Normal)
============================================================
Found 81 cases, showing 10 examples

  Sample 1/10 - Image 71
```



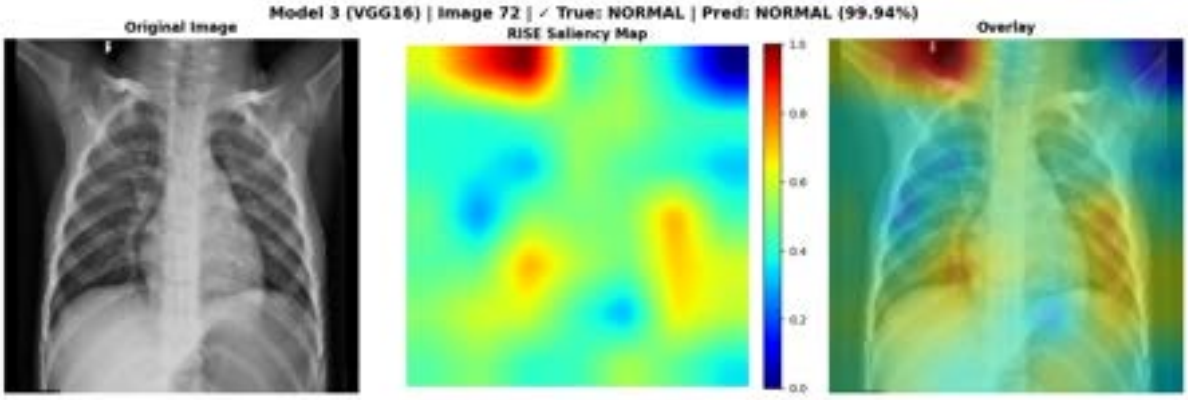

```
  Sample 2/10 - Image 2
```



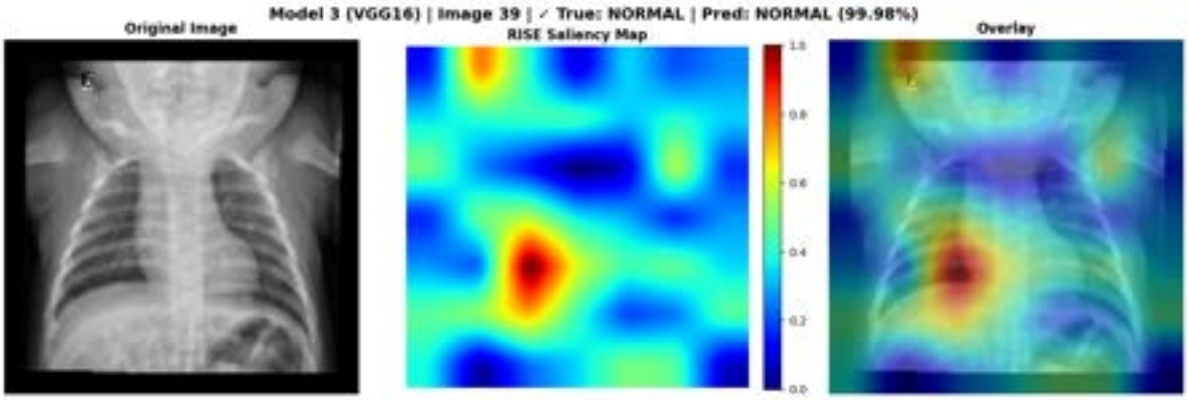

```
  Sample 3/10 - Image 54
```



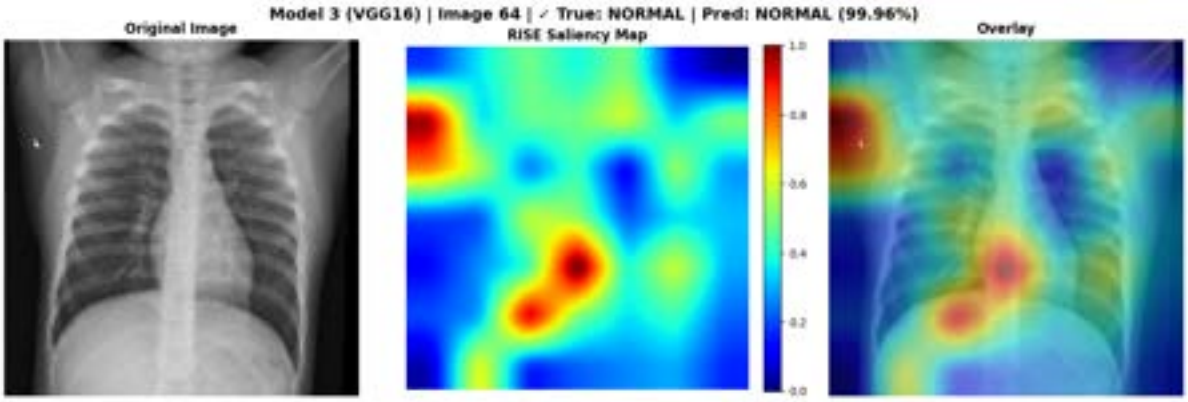
Sample 4/10 — Image 72



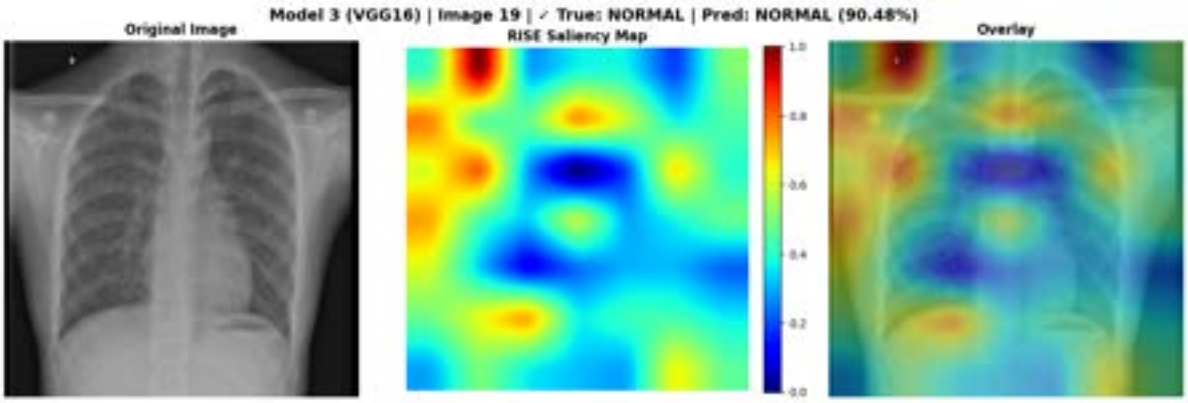
Sample 5/10 — Image 39



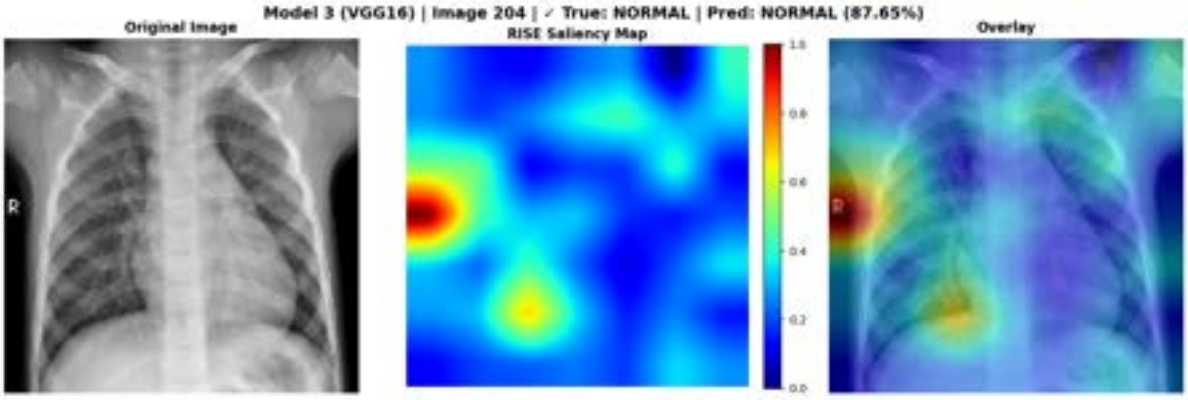
Sample 6/10 — Image 64



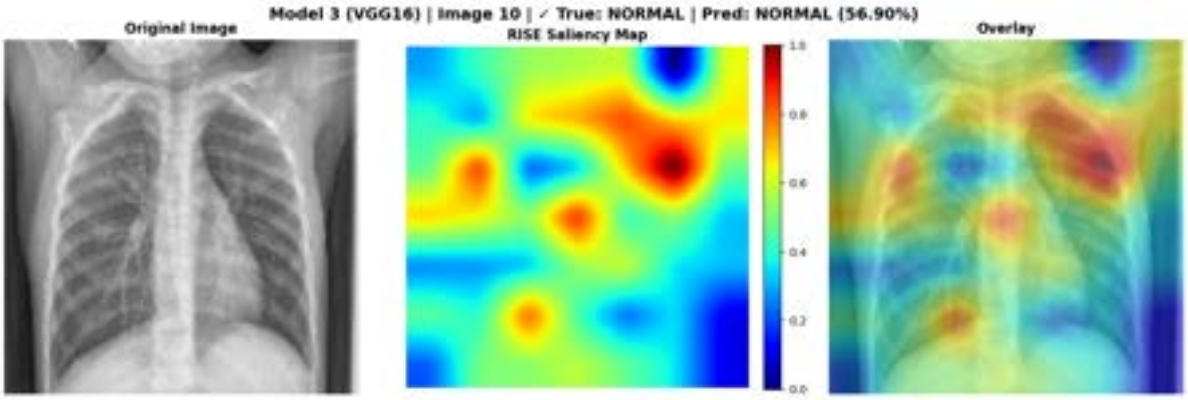
Sample 7/10 — Image 19



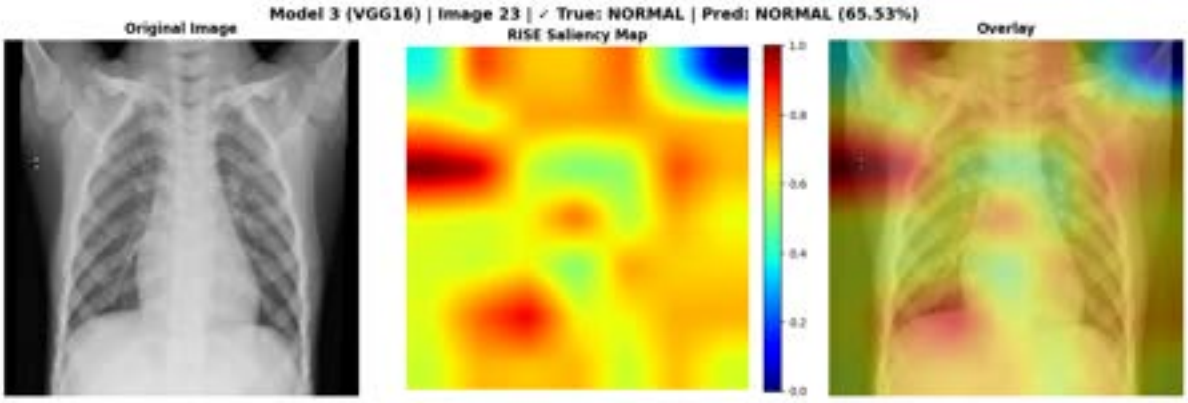
Sample 8/10 — Image 204



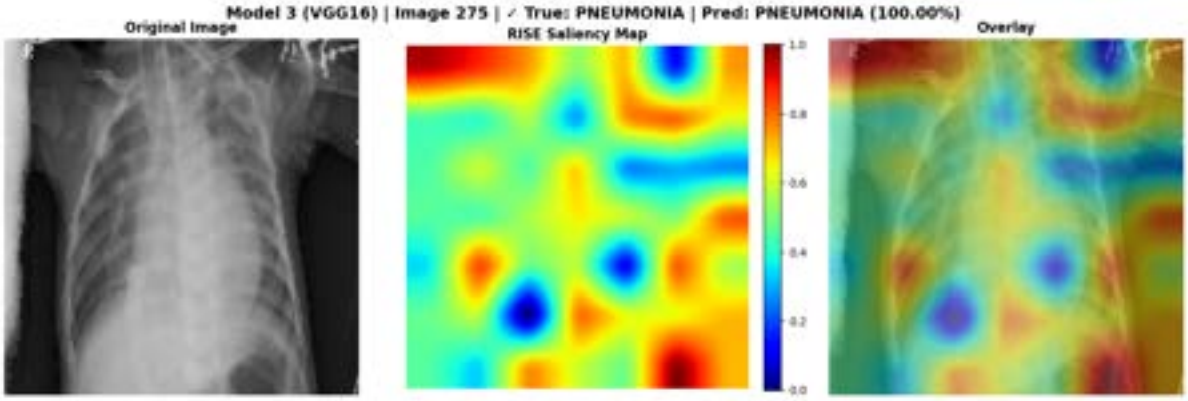
Sample 9/10 — Image 10



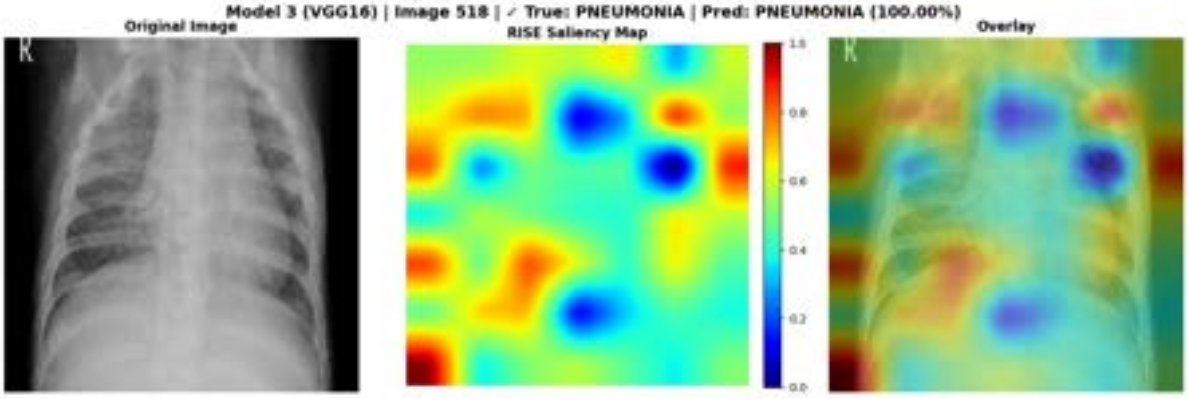
Sample 10/10 — Image 23



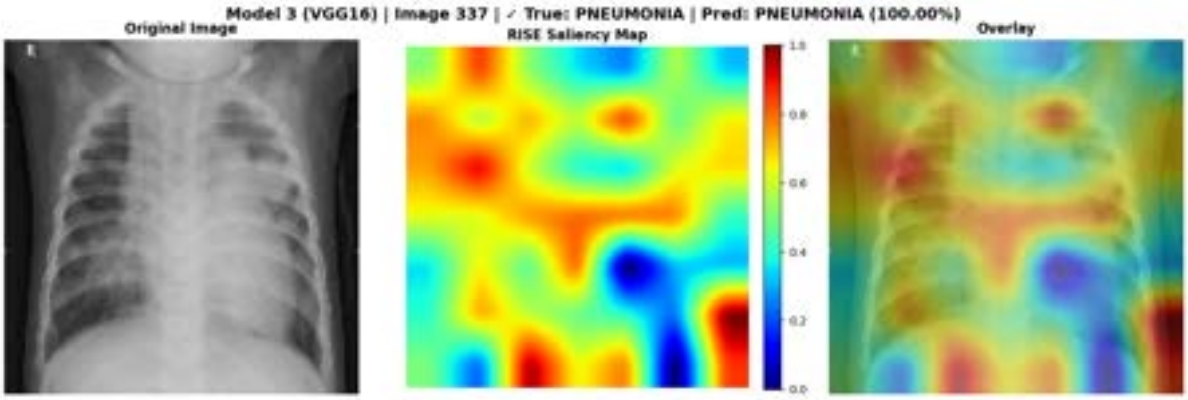

```
============================================================
✓ SCENARIO 2: True Positive (Pneumonia → Pneumonia)
============================================================
Found 388 cases, showing 10 examples

  Sample 1/10 — Image 275
```



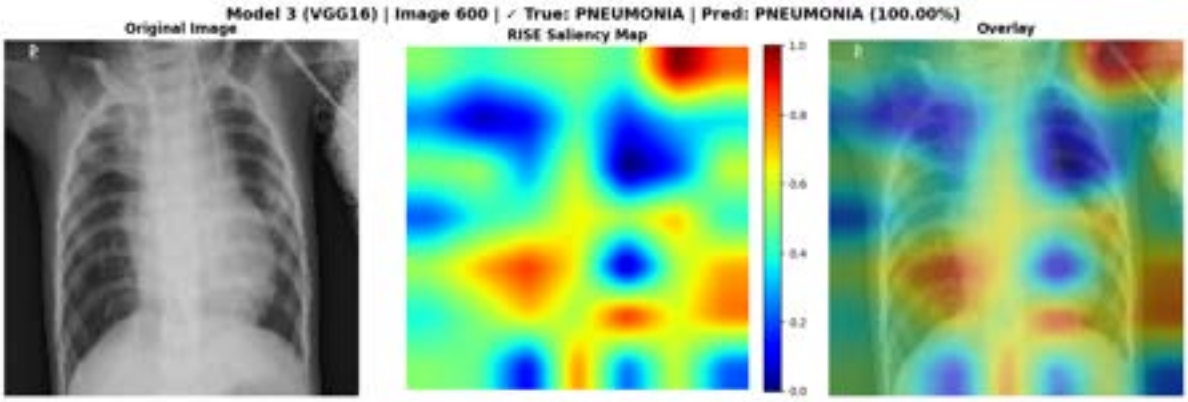
Sample 2/10 — Image 518



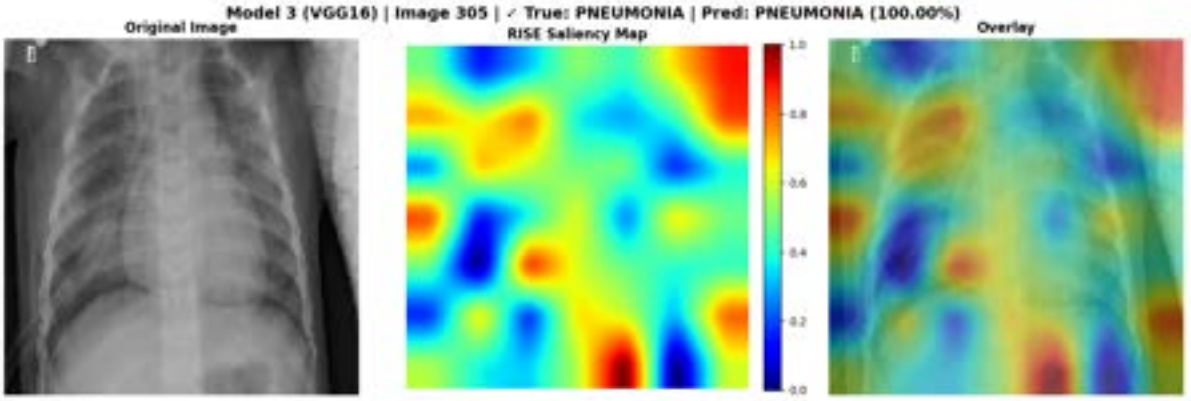
Sample 3/10 — Image 337



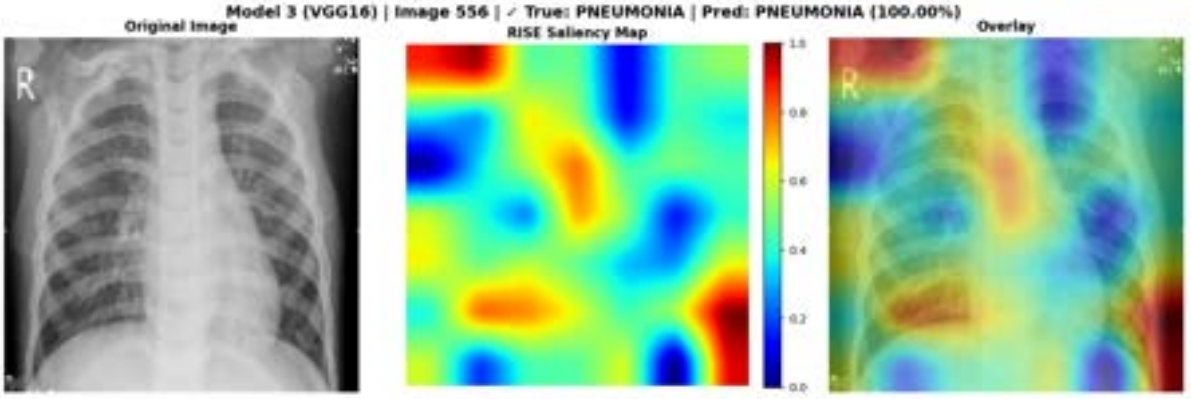
Sample 4/10 — Image 600



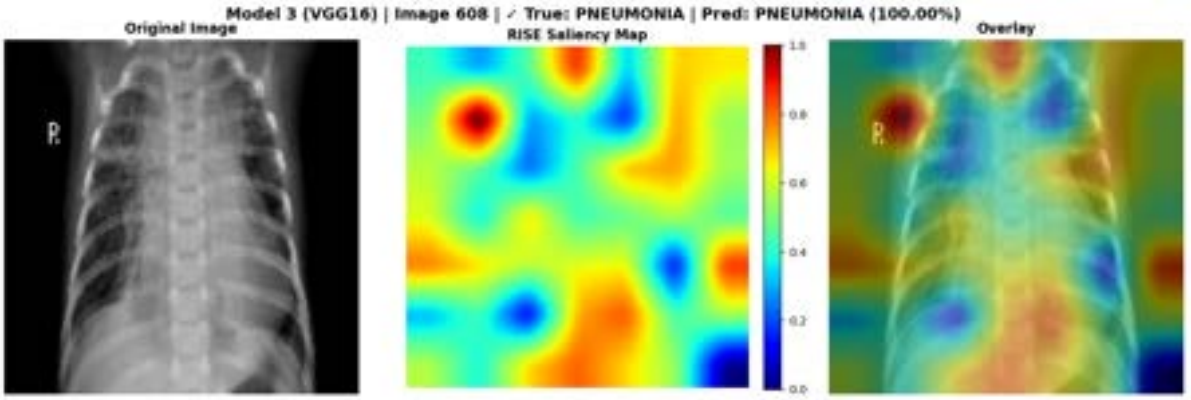
Sample 5/10 — Image 305



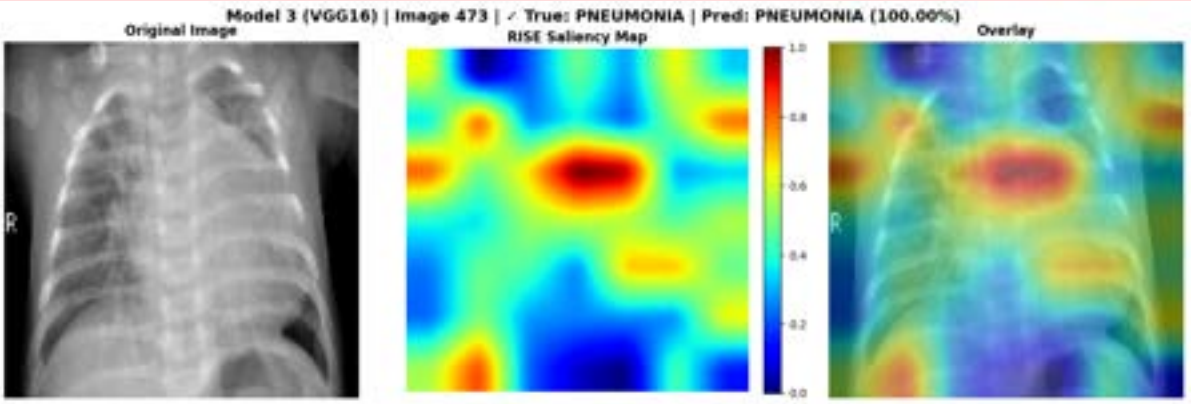
Sample 6/10 — Image 556



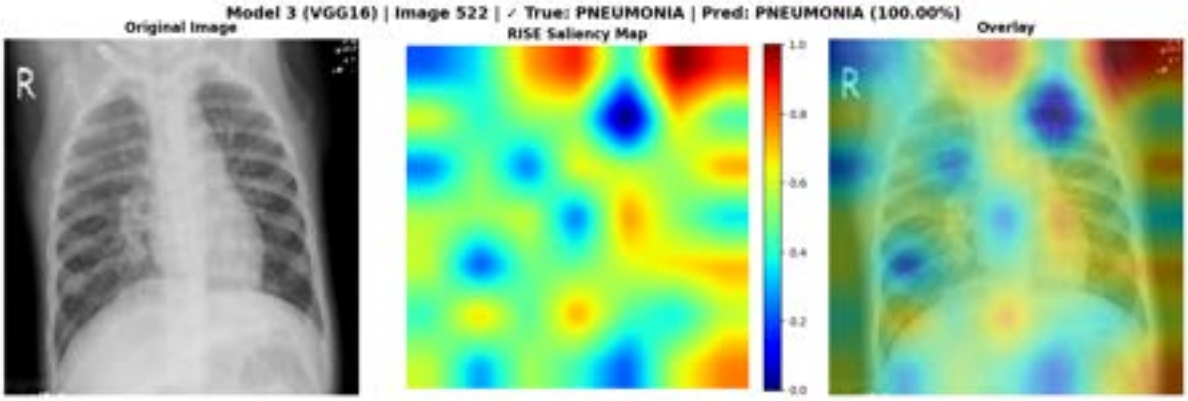
Sample 7/10 — Image 608



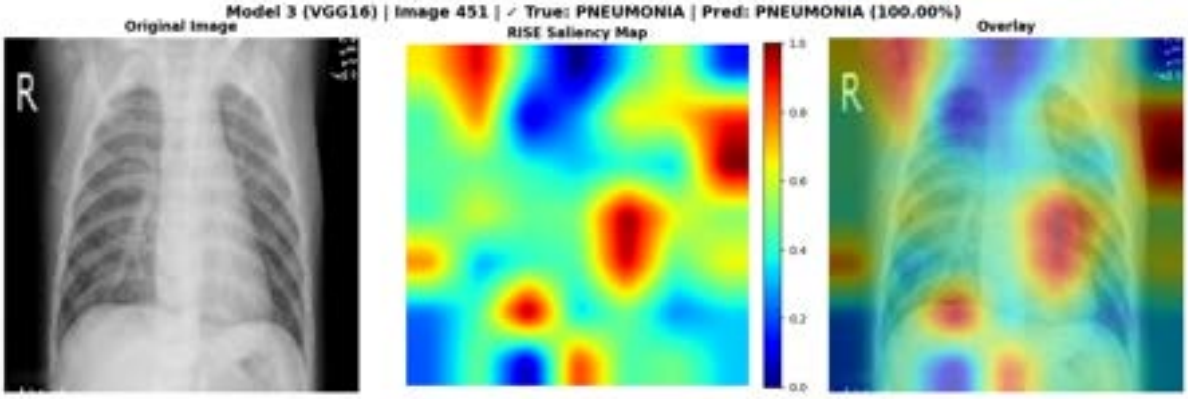
Sample 8/10 — Image 473



Sample 9/10 — Image 522

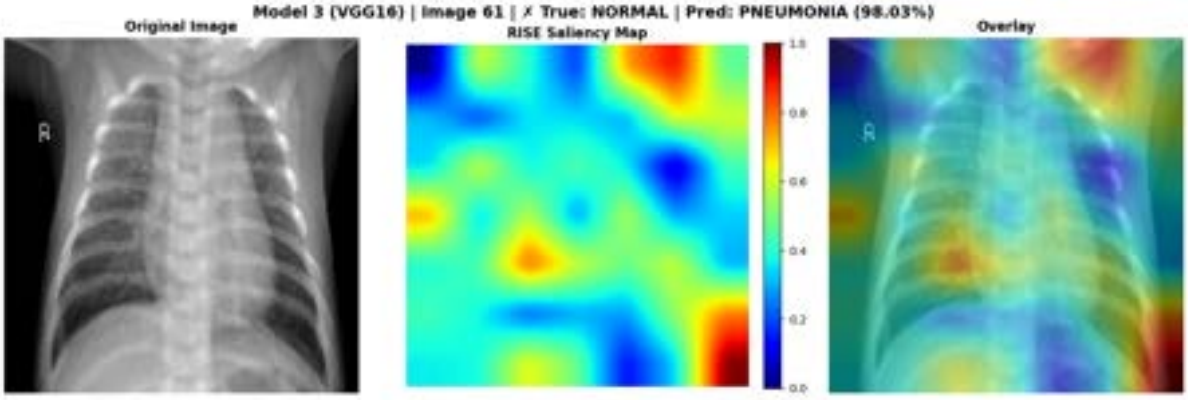


Sample 10/10 – Image 451

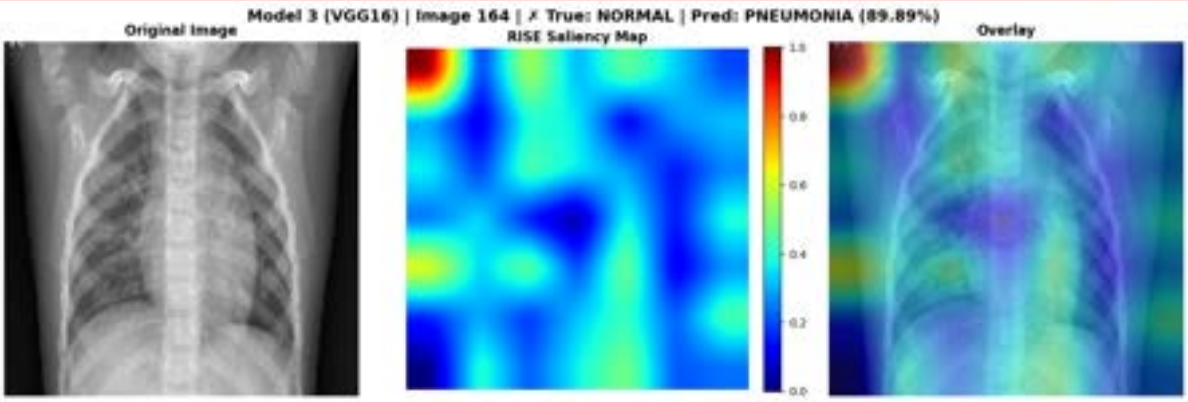



```
============================================================
✗ SCENARIO 3: False Positive (Normal → Pneumonia)
============================================================
Found 153 cases, showing 10 examples
```

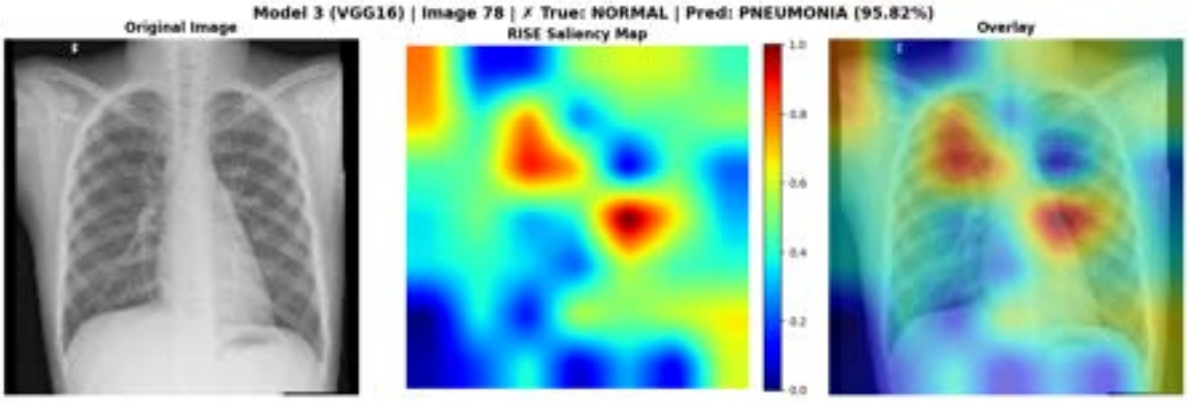
Sample 1/10 – Image 61



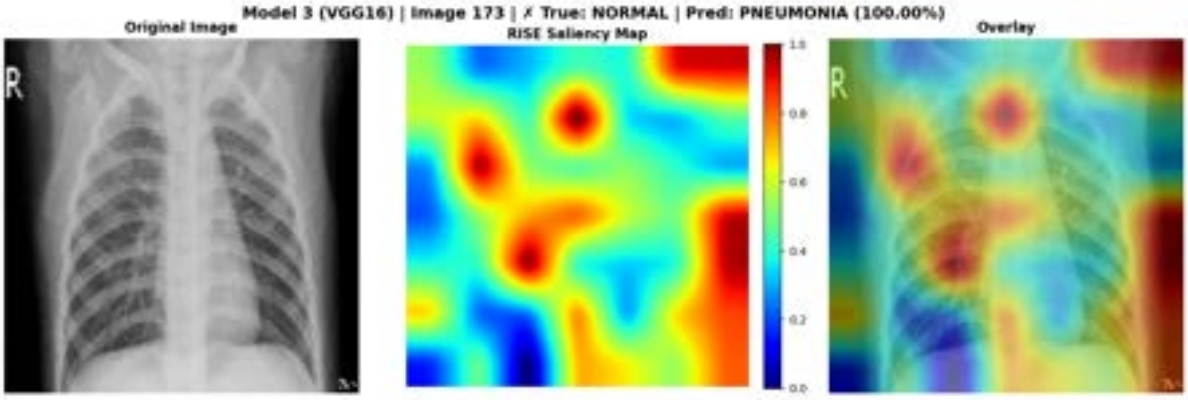
Sample 2/10 – Image 164



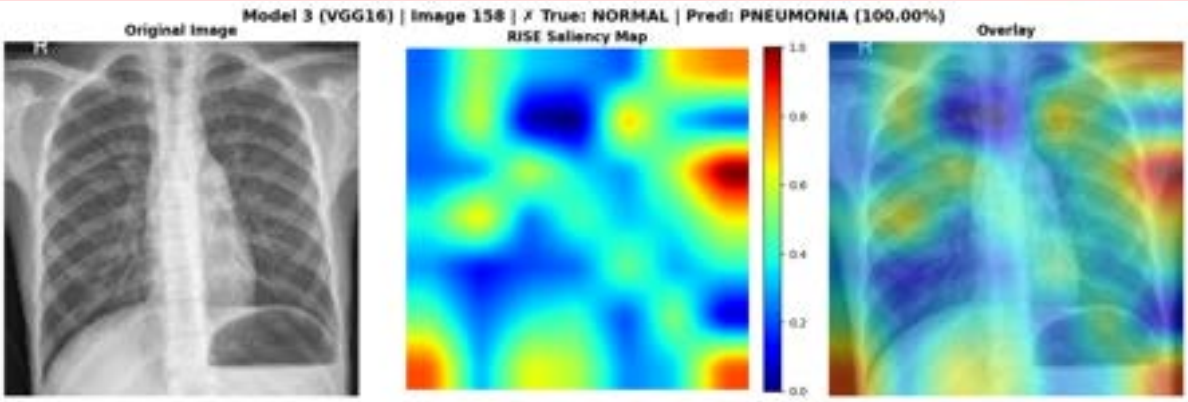
Sample 3/10 – Image 78



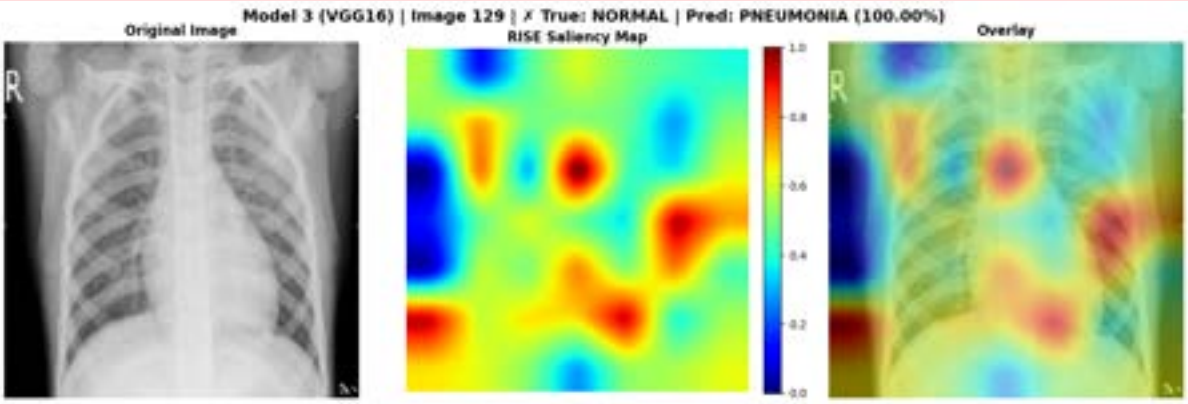
Sample 4/10 — Image 173



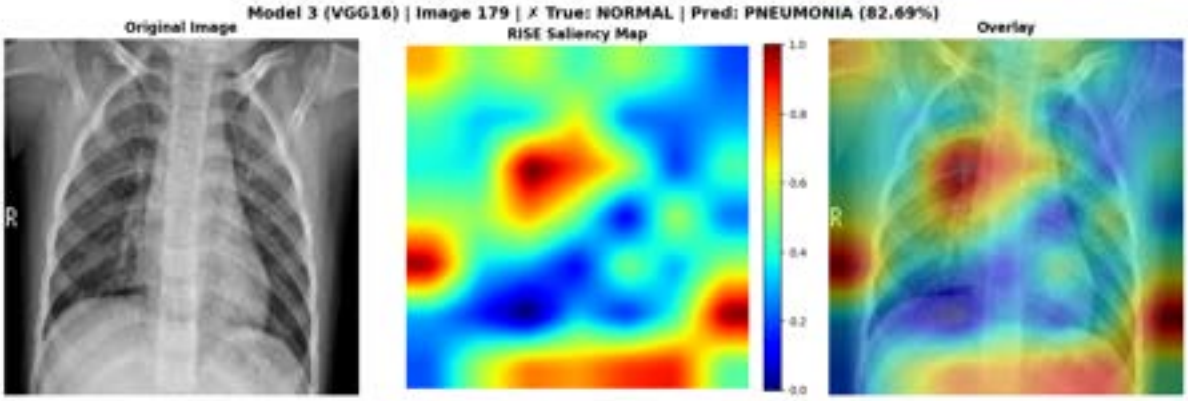
Sample 5/10 — Image 158



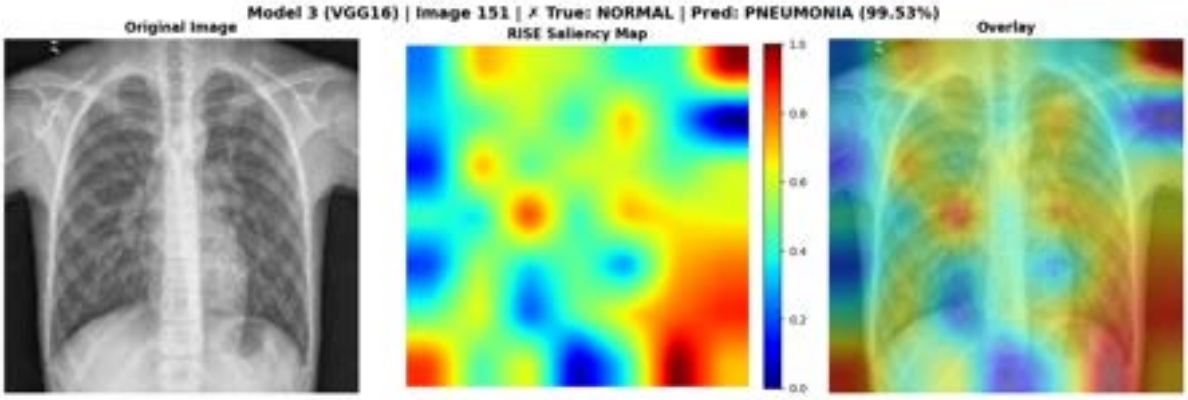
Sample 6/10 — Image 129



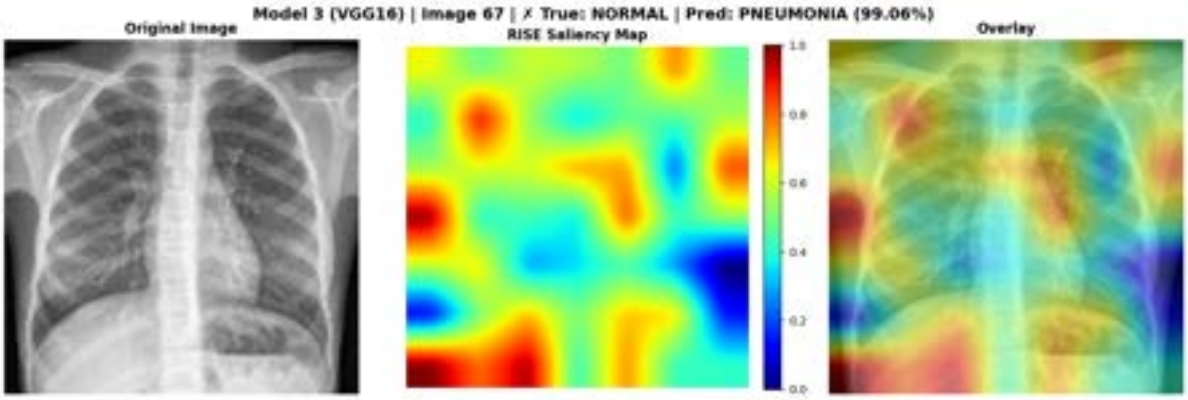
Sample 7/10 — Image 179



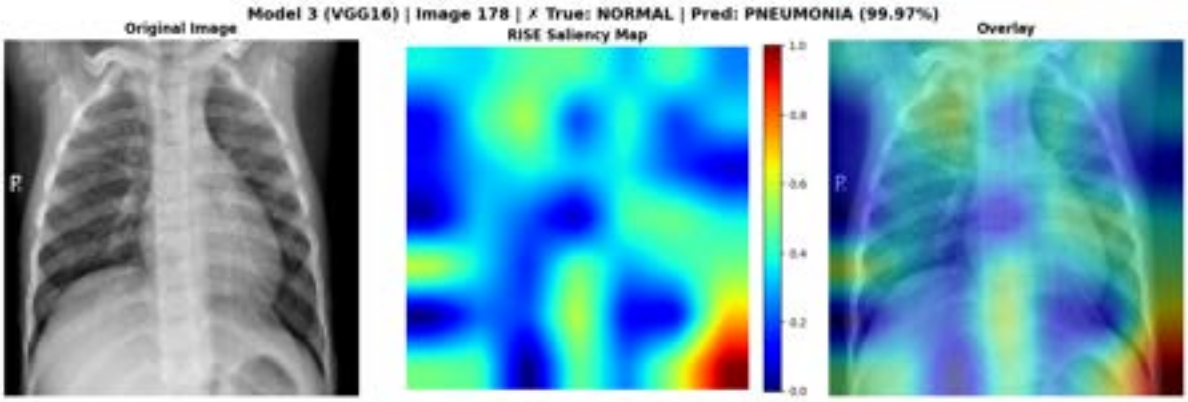
Sample 8/10 — Image 151



Sample 9/10 — Image 67

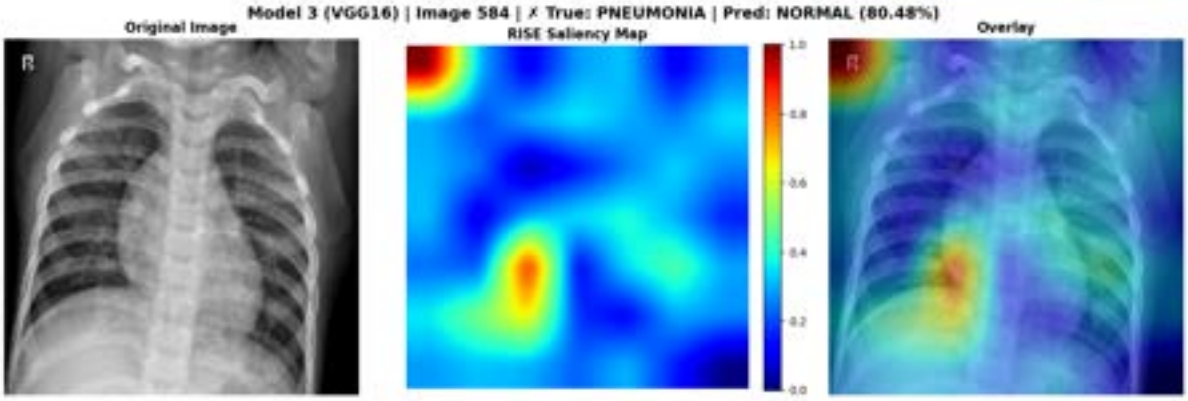


Sample 10/10 — Image 178

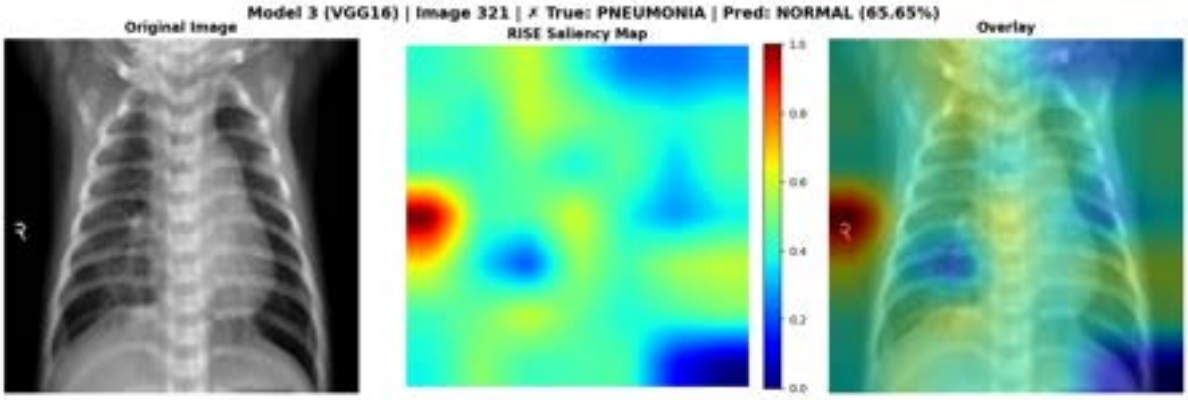



```
============================================================
✗ SCENARIO 4: False Negative (Pneumonia → Normal)
============================================================
Found 2 cases, showing 2 examples
```

Sample 1/2 — Image 584



Sample 2/2 — Image 321




```
============================================================
Model 3 (VGG16) — PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):    81 (12.98%)
✓ True Positive  (TP):   388 (62.18%)
✗ False Positive (FP):   153 (24.52%)
✗ False Negative (FN):     2 ( 0.32%)
============================================================
Overall Accuracy: 75.16%

============================================================
✅ RISE ANALYSIS COMPLETE!
============================================================

💡 KEY IMPROVEMENTS FOR SMOOTH HEATMAPS:
============================================================
✓ Using LINEAR interpolation for mask upsampling
✓ Gaussian blur (15x15 kernel) to reduce patchiness
✓ Batch processing for efficiency
✓ Proper normalization for consistent visualization
============================================================

📊 INTERPRETATION GUIDE:
============================================================
• RED/YELLOW regions = High importance for prediction
• BLUE regions       = Low importance
• Overlay shows important regions on original image
• Compare patterns across TN, TP, FP, FN scenarios
============================================================
```

In [12]:
```
# ============================================
# LIME FOR ALL 3 MODELS
# ============================================
```

```
print("\n" + "="*60)
print("LIME ANALYSIS - ALL 3 MODELS")
print("Enhanced explanations for Model 3 (VGG16)")
print("="*60)

# ============================================
# STEP 0: INSTALL LIME LIBRARY
# ============================================

print("\n" + "="*60)
print("INSTALLING LIME LIBRARY")
print("="*60)

try:
    import lime
    print("✓ LIME already installed")
except ImportError:
    print("Installing LIME library...")
    import sys
    import subprocess
    subprocess.check_call([sys.executable, "-m", "pip", "install", "lime", "
    print("✓ LIME installed successfully")

# ============================================
# STEP 1: IMPORT LIME AND SETUP
# ============================================

from lime import lime_image
from skimage.segmentation import mark_boundaries
import matplotlib.pyplot as plt
import numpy as np
import cv2

print("\n✓ LIME libraries imported successfully")

# ============================================
# STEP 2: LIME CLASSES (Standard + Enhanced)
# ============================================

class LIMEExplainer:
    """Standard LIME for Models 1 and 2"""
    def __init__(self, model, model_name="Model"):
        self.model = model
        self.model_name = model_name

        # Create LIME explainer
        self.explainer = lime_image.LimeImageExplainer()

        print(f"✓ {model_name}: LIME explainer initialized")

    def predict_fn(self, images):
        """Prediction function for LIME"""
        # Ensure images are in the right shape
        if len(images.shape) == 3:
            images = np.expand_dims(images, axis=0)

        # Get predictions
        preds = self.model.predict(images, verbose=0)

        # For binary classification, return probabilities for both classes
        if preds.shape[-1] == 1:
            preds = preds.flatten()
            # Return [prob_class_0, prob_class_1]
            return np.column_stack([1 - preds, preds])
```

```
        else:
            return preds

    def explain_instance(self, image, num_samples=1000, top_labels=2):
        """
        Generate LIME explanation for an image

        Args:
            image: Input image (150, 150, 3)
            num_samples: Number of samples for LIME
            top_labels: Number of top labels to explain
        """
        try:
            # Generate explanation
            explanation = self.explainer.explain_instance(
                image,
                self.predict_fn,
                top_labels=top_labels,
                hide_color=0,
                num_samples=num_samples,
                batch_size=32
            )

            return explanation

        except Exception as e:
            print(f"❌ Error generating LIME explanation for {self.model_nar
            return None


class EnhancedLIMEExplainer(LIMEExplainer):
    """Enhanced LIME specifically for Model 3 (VGG16) - Better pneumonia vis

    def explain_instance(self, image, num_samples=2000, top_labels=2, num_fe
        """
        Enhanced LIME explanation with improvements for pneumonia detection

        Args:
            image: Input image (150, 150, 3)
            num_samples: Number of samples (MORE samples = better accuracy)
            top_labels: Number of top labels to explain
            num_features: Number of superpixels to highlight
        """
        try:
            # IMPROVEMENT 1: Use more samples for better accuracy
            # IMPROVEMENT 2: Show more features (superpixels)
            explanation = self.explainer.explain_instance(
                image,
                self.predict_fn,
                top_labels=top_labels,
                hide_color=0,
                num_samples=num_samples,  # More samples = better explanatio
                batch_size=32,
                segmentation_fn=None  # Use default segmentation
            )

            return explanation

        except Exception as e:
            print(f"❌ Error generating Enhanced LIME explanation for {self
            return None


# ============================================
```

```
# STEP 3: INITIALIZE LIME FOR ALL MODELS
# ============================================

print("\n" + "="*60)
print("INITIALIZING LIME EXPLAINERS")
print("="*60)

# Initialize LIME objects only for models that exist
lime_model1 = None
lime_model2 = None
lime_model3 = None

# Check if models exist
print("\nChecking model availability...")
print(f"model1 exists: {'model1' in dir()}")
print(f"model2 exists: {'model2' in dir()}")
print(f"model3 exists: {'model3' in dir()}")

# For Model 1 - Standard LIME
if 'model1' in dir():
    try:
        print(f"\nCreating LIME explainer for Model 1 (Simple CNN)...")
        lime_model1 = LIMEExplainer(model1, model_name="Model 1 (Simple CNN)
        print(f"  ✓ LIME explainer created for Model 1")
    except Exception as e:
        print(f"  ✗ Error creating LIME explainer for Model 1: {e}")

# For Model 2 - Standard LIME
if 'model2' in dir():
    try:
        print(f"\nCreating LIME explainer for Model 2 (ResNet)...")
        lime_model2 = LIMEExplainer(model2, model_name="Model 2 (ResNet)")
        print(f"  ✓ LIME explainer created for Model 2")
    except Exception as e:
        print(f"  ✗ Error creating LIME explainer for Model 2: {e}")

# For Model 3 - ENHANCED LIME
if 'model3' in dir():
    try:
        print(f"\nCreating ENHANCED LIME explainer for Model 3 (VGG16)...")
        lime_model3 = EnhancedLIMEExplainer(model3, model_name="Model 3 (VG
        print(f"  ✓ ENHANCED LIME explainer created for Model 3")
        print(f"  → Using more samples and features for better explanations'
    except Exception as e:
        print(f"  ✗ Error creating LIME explainer for Model 3: {e}")

print("\n" + "="*60)
print("LIME INITIALIZATION SUMMARY")
print("="*60)
print(f"Model 1 LIME: {'✓ Ready (Standard)' if lime_model1 else '✗ Not avai
print(f"Model 2 LIME: {'✓ Ready (Standard)' if lime_model2 else '✗ Not avai
print(f"Model 3 LIME: {'✓ Ready (ENHANCED)' if lime_model3 else '✗ Not avai

# ============================================
# STEP 4: VISUALIZATION FUNCTION
# ============================================

def visualize_lime_triple(img, true_label, pred_label, pred_prob, lime_expla
                          model_name, image_id=None, is_enhanced=False):
    """Visualize LIME: Original, Positive Features, Negative Features"""

    # Generate LIME explanation
    print(f"  Generating LIME explanation for image {image_id}...", end=" ")
```

```
    if is_enhanced:
        # Enhanced version with more samples and features
        explanation = lime_explainer.explain_instance(
            img,
            num_samples=2000,  # More samples for Model 3
            num_features=10
        )
    else:
        # Standard version
        explanation = lime_explainer.explain_instance(
            img,
            num_samples=1000,
            top_labels=2
        )

    if explanation is None:
        print("Failed!")
        return

    print("Done!")

    # Get the predicted label
    pred_label_int = int(pred_label)

    # Create figure with 3 subplots
    fig = plt.figure(figsize=(15, 5))

    # 1. Original Image
    plt.subplot(1, 3, 1)
    plt.title("Original Image", fontsize=12, fontweight='bold')
    plt.axis("off")
    if len(img.shape) == 2:
        plt.imshow(img, cmap='gray', interpolation='bilinear')
    else:
        plt.imshow(img, interpolation='bilinear')

    # 2. Positive Features (support prediction)
    plt.subplot(1, 3, 2)
    plt.title("Positive Features (Support Prediction)", fontsize=12, fontwei
    plt.axis("off")

    temp_pos, mask_pos = explanation.get_image_and_mask(
        pred_label_int,
        positive_only=True,
        num_features=5 if not is_enhanced else 10,  # More features for enha
        hide_rest=False
    )
    plt.imshow(mark_boundaries(temp_pos, mask_pos), interpolation='bilinear

    # 3. Negative Features (contradict prediction)
    plt.subplot(1, 3, 3)
    plt.title("Negative Features (Contradict Prediction)", fontsize=12, font
    plt.axis("off")

    temp_neg, mask_neg = explanation.get_image_and_mask(
        pred_label_int,
        positive_only=False,
        num_features=5 if not is_enhanced else 10,  # More features for enha
        hide_rest=False,
        negative_only=True
    )
    plt.imshow(mark_boundaries(temp_neg, mask_neg), interpolation='bilinear

    # Create title
```

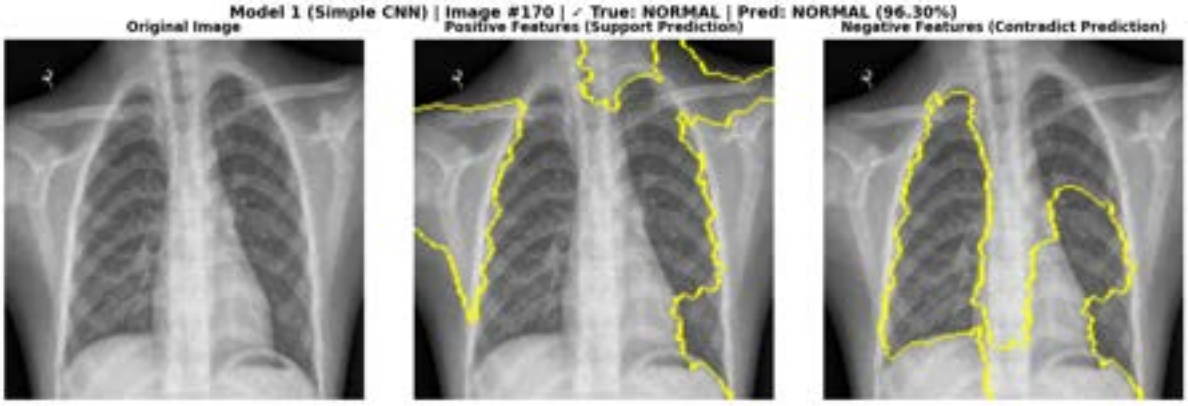
```
    true_class = 'NORMAL' if true_label == 0 else 'PNEUMONIA'
    pred_class = 'NORMAL' if pred_label == 0 else 'PNEUMONIA'
    confidence = pred_prob if pred_label == 1 else (1 - pred_prob)
    match_symbol = "✓" if true_label == pred_label else "✗"

    if image_id is not None:
        title = f"{model_name} | Image #{image_id} | {match_symbol} True: {
    else:
        title = f"{model_name} | {match_symbol} True: {true_class} | Pred: 

    fig.suptitle(title, fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()


# ============================================
# STEP 5: ANALYZE ALL SCENARIOS FOR ALL MODELS
# ============================================

def analyze_model_lime(model_name, lime_explainer, y_pred, y_pred_prob, num_
    """Analyze one model with LIME for all 4 scenarios"""

    if lime_explainer is None:
        print(f"\n❌ {model_name}: LIME explainer not available. Skipping a
        return

    print("\n" + "="*60)
    print(f"LIME ANALYSIS: {model_name}")
    print("="*60)

    # Define scenarios
    scenarios = [
        {
            'name': 'SCENARIO 1: True Negative (TN) - Normal→Normal ✓',
            'mask': (y_test == 0) & (y_pred == 0),
            'short': 'TN'
        },
        {
            'name': 'SCENARIO 2: True Positive (TP) - Pneumonia→Pneumonia ✓
            'mask': (y_test == 1) & (y_pred == 1),
            'short': 'TP'
        },
        {
            'name': 'SCENARIO 3: False Positive (FP) - Normal→Pneumonia ✗',
            'mask': (y_test == 0) & (y_pred == 1),
            'short': 'FP'
        },
        {
            'name': 'SCENARIO 4: False Negative (FN) - Pneumonia→Normal ✗',
            'mask': (y_test == 1) & (y_pred == 0),
            'short': 'FN'
        }
    ]

    # Visualize each scenario
    np.random.seed(42)
    for scenario in scenarios:
        indices = np.where(scenario['mask'])[0]

        if len(indices) == 0:
            print(f"\n❌ {scenario['name']}: No cases found")
            continue

        print(f"\n{'='*60}")
```

```
        print(f"{scenario['name']}")
        print(f"{'='*60}")
        print(f"Found {len(indices)} cases, showing {min(num_samples, len(in

        # Select samples
        selected_indices = np.random.choice(indices, min(num_samples, len(in

        for i, idx in enumerate(selected_indices, 1):
            img = X_test_scaled[idx]
            true_label = y_test[idx]
            pred_prob = y_pred_prob[idx][0] if isinstance(y_pred_prob[idx],
            pred_label = y_pred[idx]

            print(f"Sample {i}/{len(selected_indices)} - Image Index: {idx}"
            visualize_lime_triple(img, true_label, pred_label, pred_prob,
                                  lime_explainer, model_name, image_id=idx,
                                  is_enhanced=is_enhanced)

    # Print summary
    print(f"\n{'='*60}")
    print(f"{model_name} - PREDICTION SUMMARY")
    print(f"{'='*60}")

    tn = np.sum((y_test == 0) & (y_pred == 0))
    tp = np.sum((y_test == 1) & (y_pred == 1))
    fp = np.sum((y_test == 0) & (y_pred == 1))
    fn = np.sum((y_test == 1) & (y_pred == 0))
    total = len(y_test)

    print(f"\nTest Set Breakdown (Total: {total} samples):")
    print(f"{'='*60}")
    print(f"✓ True Negative  (TN):  {tn:4d} ({tn/total*100:5.2f}%)")
    print(f"✓ True Positive  (TP):  {tp:4d} ({tp/total*100:5.2f}%)")
    print(f"✗ False Positive (FP):  {fp:4d} ({fp/total*100:5.2f}%)")
    print(f"✗ False Negative (FN):  {fn:4d} ({fn/total*100:5.2f}%)")
    print(f"{'='*60}")
    print(f"Overall Accuracy: {(tn + tp)/total*100:.2f}%")


# ============================================
# STEP 6: RUN ANALYSIS FOR ALL 3 MODELS
# ============================================

print("\n" + "="*60)
print("STARTING LIME ANALYSIS FOR ALL MODELS")
print("Generating 10 samples per scenario (TN, TP, FP, FN)")
print("Model 3 uses ENHANCED explanation generation")
print("="*60)

# Check if y_pred variables exist
print("\nChecking prediction variables...")
print(f"y_pred1 exists: {'y_pred1' in dir()}")
print(f"y_pred2 exists: {'y_pred2' in dir()}")
print(f"y_pred3 exists: {'y_pred3' in dir()}")
print(f"y_pred1_prob exists: {'y_pred1_prob' in dir()}")
print(f"y_pred2_prob exists: {'y_pred2_prob' in dir()}")
print(f"y_pred3_prob exists: {'y_pred3_prob' in dir()}")

# Model 1: Simple CNN (only if everything is available)
if lime_model1 and 'y_pred1' in dir() and 'y_pred1_prob' in dir():
    analyze_model_lime(
        model_name="Model 1 (Simple CNN)",
        lime_explainer=lime_model1,
        y_pred=y_pred1,
```

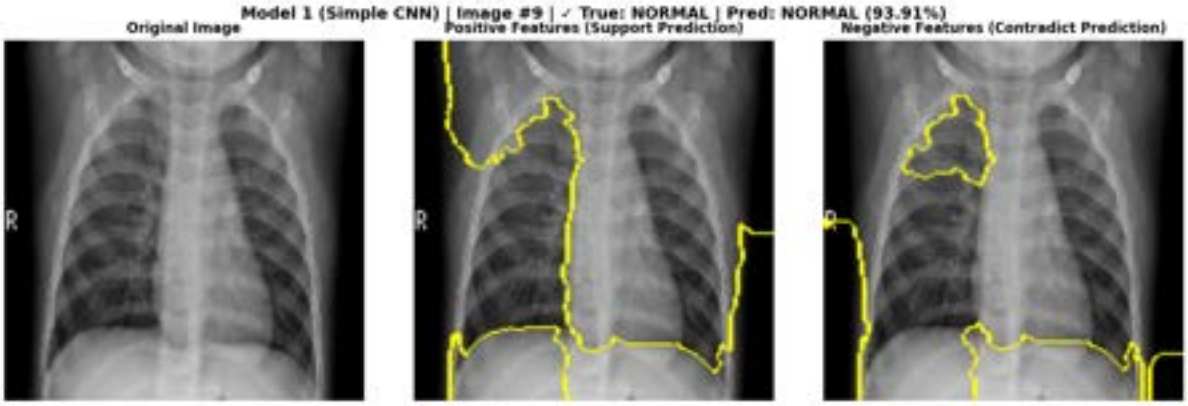
```
        y_pred_prob=y_pred1_prob,
        num_samples=10,
        is_enhanced=False
    )
else:
    print("\n❌ Skipping Model 1: Missing LIME explainer or predictions")

# Model 2: ResNet (only if everything is available)
if lime_model2 and 'y_pred2' in dir() and 'y_pred2_prob' in dir():
    analyze_model_lime(
        model_name="Model 2 (ResNet)",
        lime_explainer=lime_model2,
        y_pred=y_pred2,
        y_pred_prob=y_pred2_prob,
        num_samples=10,
        is_enhanced=False
    )
else:
    print("\n❌ Skipping Model 2: Missing LIME explainer or predictions")

# Model 3: VGG16 with ENHANCED LIME (only if everything is available)
if lime_model3 and 'y_pred3' in dir() and 'y_pred3_prob' in dir():
    analyze_model_lime(
        model_name="Model 3 (VGG16) - ENHANCED",
        lime_explainer=lime_model3,
        y_pred=y_pred3,
        y_pred_prob=y_pred3_prob,
        num_samples=10,
        is_enhanced=True
    )
else:
    print("\n❌ Skipping Model 3: Missing LIME explainer or predictions")

print("\n" + "="*60)
print("LIME ANALYSIS COMPLETE!")
print("="*60)
print("\nMODEL 3 ENHANCEMENTS:")
print("  ✓ More Samples (2000 vs 1000) - Better accuracy")
print("  ✓ More Features (10 vs 5) - More detailed explanations")
print("  ✓ Superpixel Segmentation - Region-based analysis")
print("  ✓ Positive/Negative Feature Separation - Clear interpretation")
print("="*60)
```

```
============================================================
LIME ANALYSIS - ALL 3 MODELS
Enhanced explanations for Model 3 (VGG16)
============================================================

============================================================
INSTALLING LIME LIBRARY
============================================================
Installing LIME library...
✓ LIME installed successfully

✓ LIME libraries imported successfully

============================================================
INITIALIZING LIME EXPLAINERS
============================================================

Checking model availability...
model1 exists: True
model2 exists: True
model3 exists: True

Creating LIME explainer for Model 1 (Simple CNN)...
✓ Model 1 (Simple CNN): LIME explainer initialized
  ✓ LIME explainer created for Model 1

Creating LIME explainer for Model 2 (ResNet)...
✓ Model 2 (ResNet): LIME explainer initialized
  ✓ LIME explainer created for Model 2

Creating ENHANCED LIME explainer for Model 3 (VGG16)...
✓ Model 3 (VGG16) - Enhanced: LIME explainer initialized
  ✓ ENHANCED LIME explainer created for Model 3
  → Using more samples and features for better explanations

============================================================
LIME INITIALIZATION SUMMARY
============================================================
Model 1 LIME: ✓ Ready (Standard)
Model 2 LIME: ✓ Ready (Standard)
Model 3 LIME: ✓ Ready (ENHANCED)

============================================================
STARTING LIME ANALYSIS FOR ALL MODELS
Generating 10 samples per scenario (TN, TP, FP, FN)
Model 3 uses ENHANCED explanation generation
============================================================

Checking prediction variables...
y_pred1 exists: True
y_pred2 exists: True
y_pred3 exists: True
y_pred1_prob exists: True
y_pred2_prob exists: True
y_pred3_prob exists: True

============================================================
LIME ANALYSIS: Model 1 (Simple CNN)
============================================================

============================================================
SCENARIO 1: True Negative (TN) - Normal→Normal ✓
============================================================
Found 115 cases, showing 10 examples...
```

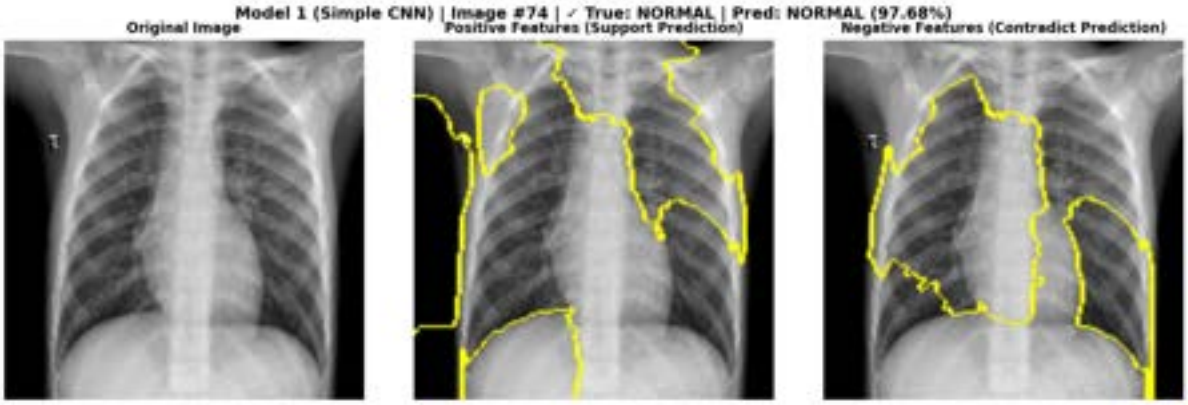
```
Sample 1/10 - Image Index: 170
  Generating LIME explanation for image 170...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



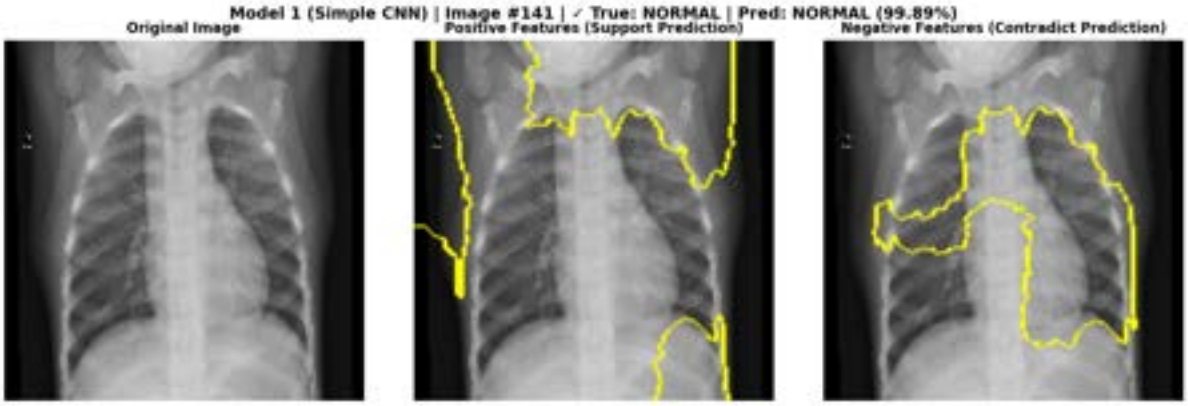

```
Sample 2/10 - Image Index: 9
  Generating LIME explanation for image 9...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



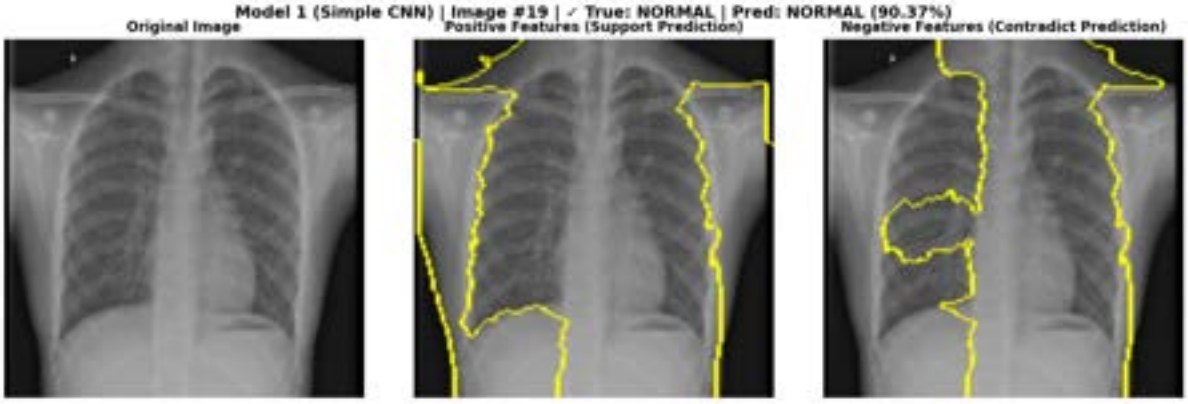

```
Sample 3/10 - Image Index: 74
  Generating LIME explanation for image 74...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



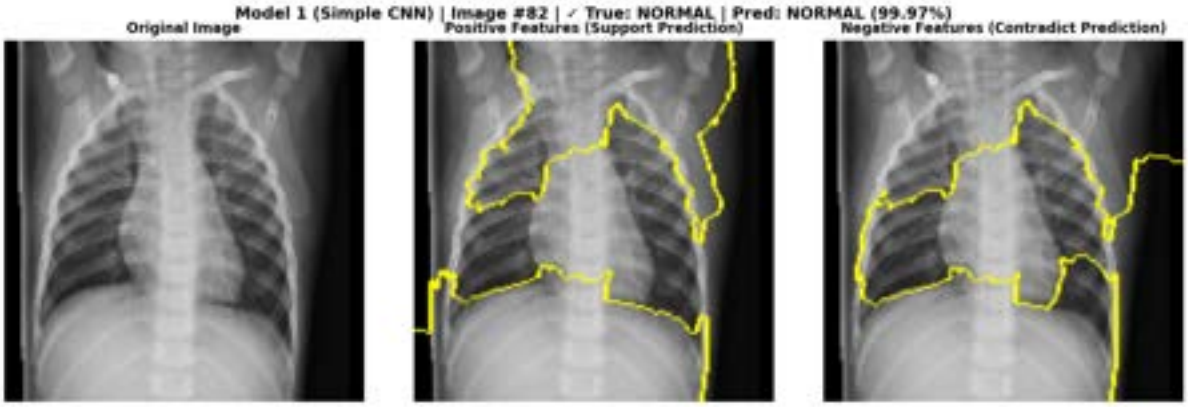

```
Sample 4/10 - Image Index: 141
  Generating LIME explanation for image 141...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



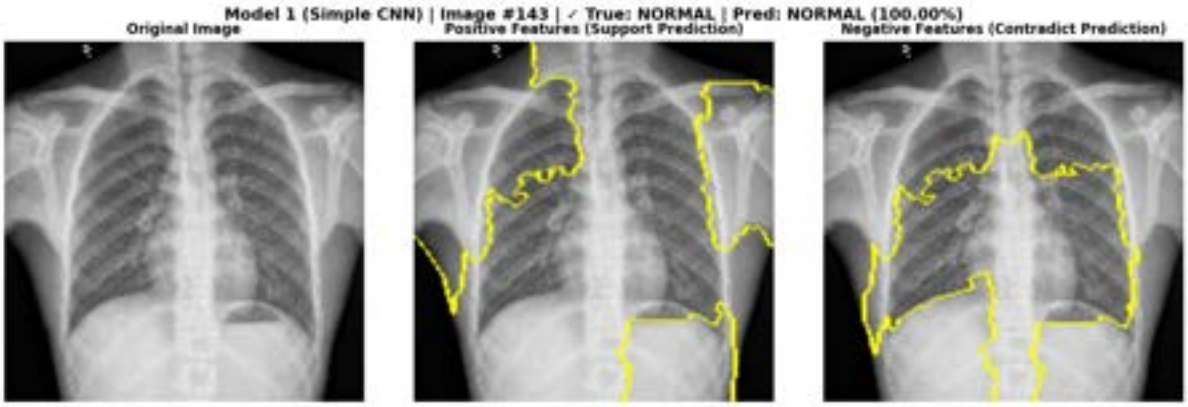

```
Sample 5/10 - Image Index: 19
  Generating LIME explanation for image 19...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```




```
Sample 6/10 - Image Index: 82
  Generating LIME explanation for image 82...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

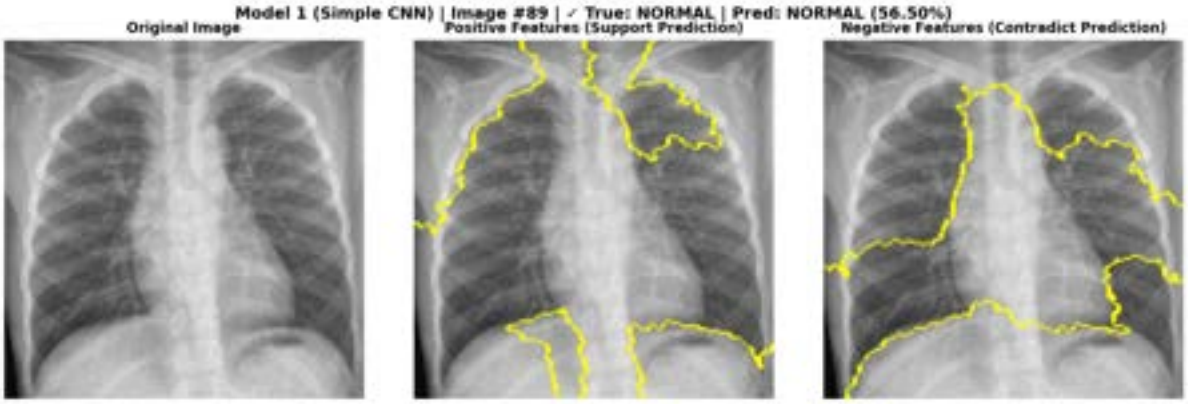



```
Sample 7/10 - Image Index: 143
  Generating LIME explanation for image 143...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



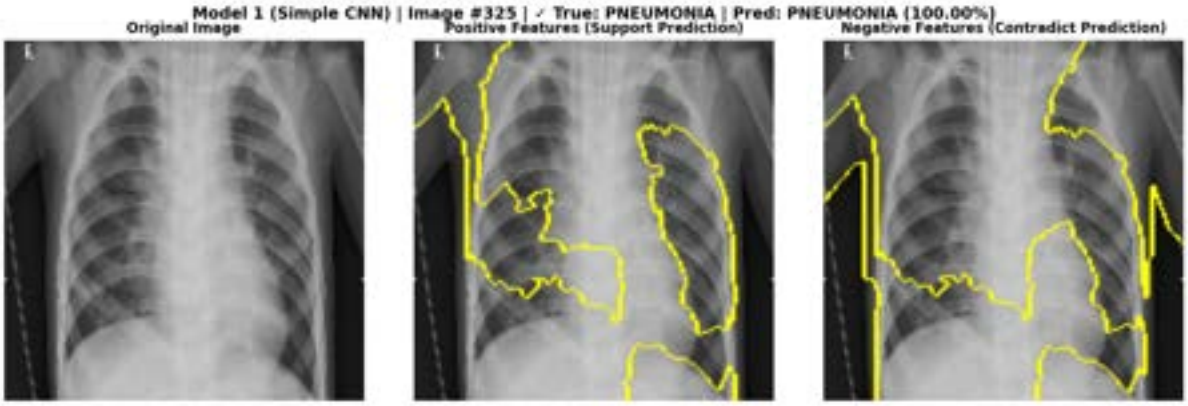
```
Sample 8/10 - Image Index: 137
  Generating LIME explanation for image 137...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```


```
Sample 9/10 - Image Index: 89
  Generating LIME explanation for image 89...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

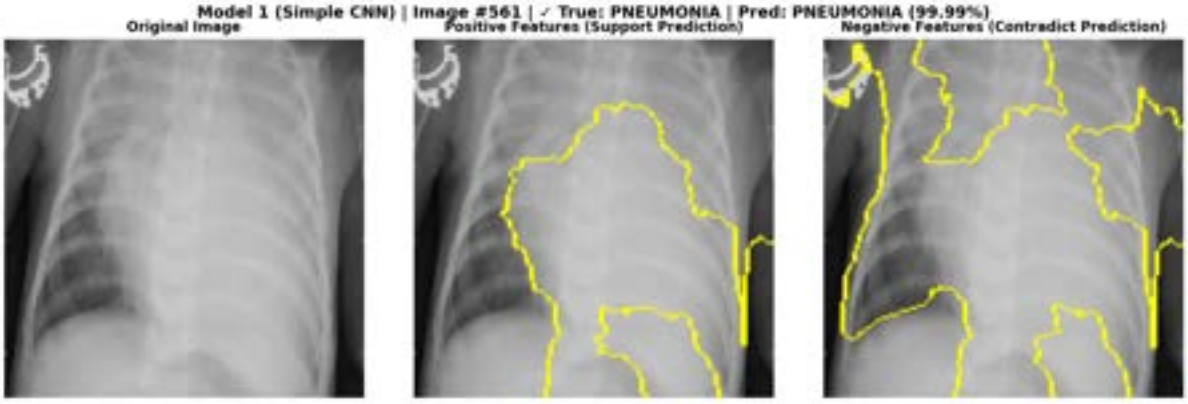



```
Sample 10/10 - Image Index: 20
  Generating LIME explanation for image 20...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

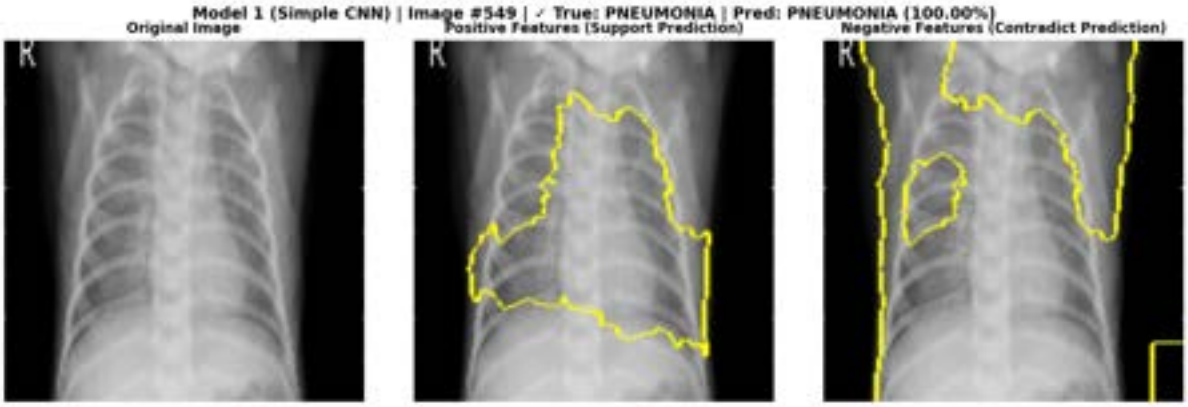

```
============================================================
SCENARIO 2: True Positive (TP) - Pneumonia→Pneumonia ✓
============================================================
Found 387 cases, showing 10 examples...

Sample 1/10 - Image Index: 325
  Generating LIME explanation for image 325...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



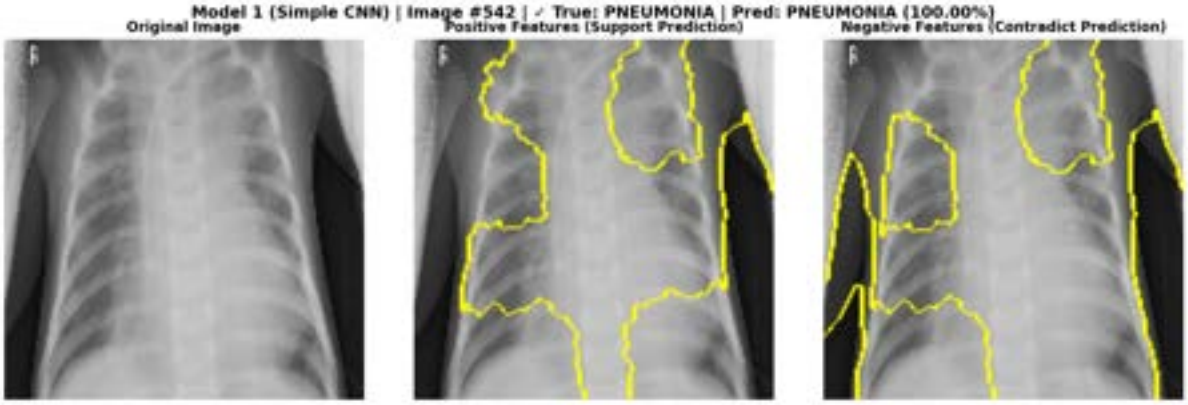

```
Sample 2/10 - Image Index: 561
  Generating LIME explanation for image 561...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



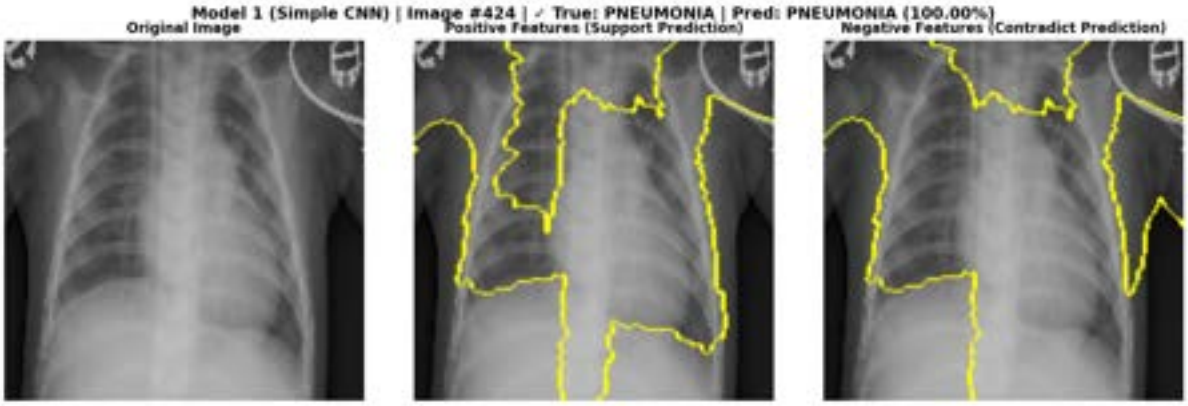

```
Sample 3/10 - Image Index: 549
  Generating LIME explanation for image 549...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



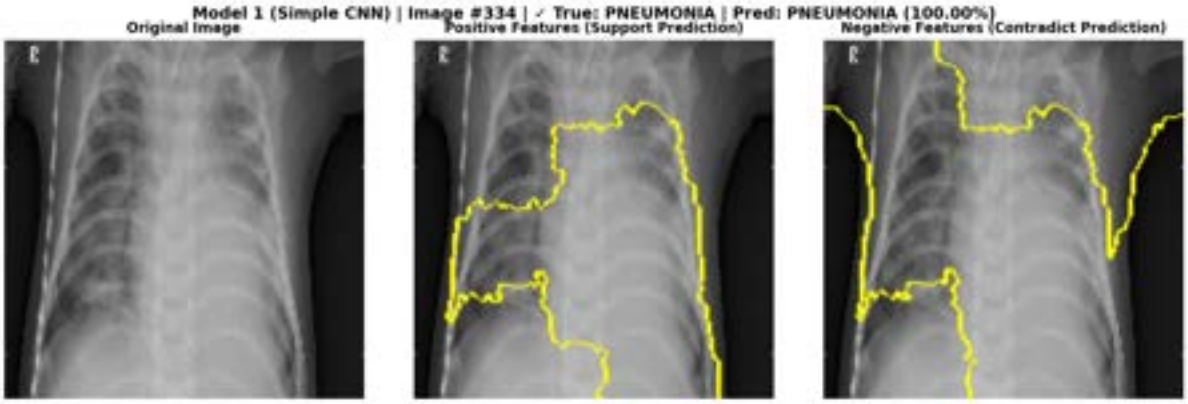

```
Sample 4/10 - Image Index: 542
  Generating LIME explanation for image 542...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



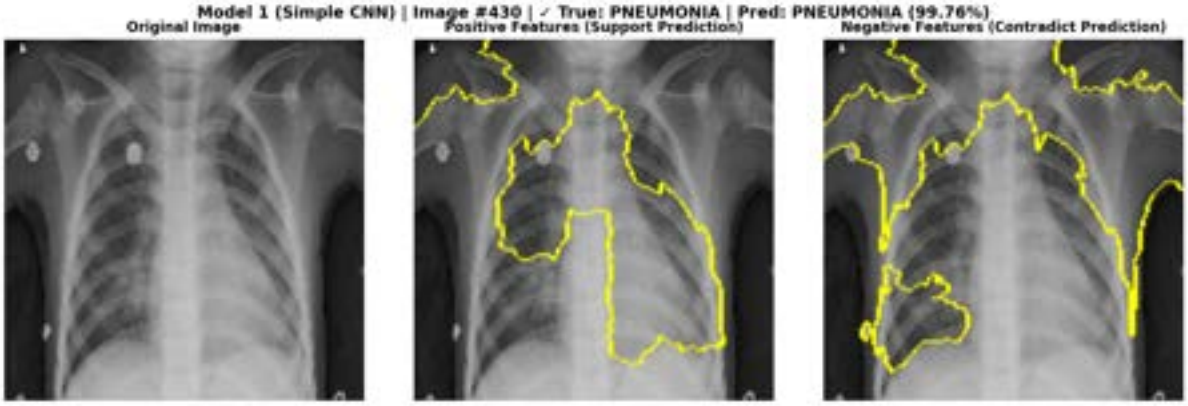
```
Sample 5/10 - Image Index: 424
  Generating LIME explanation for image 424...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



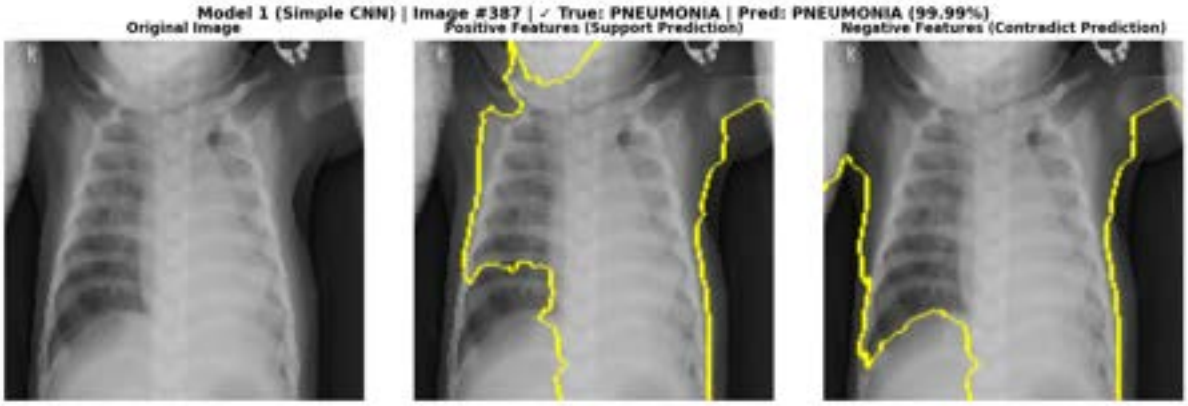

```
Sample 6/10 - Image Index: 334
  Generating LIME explanation for image 334...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



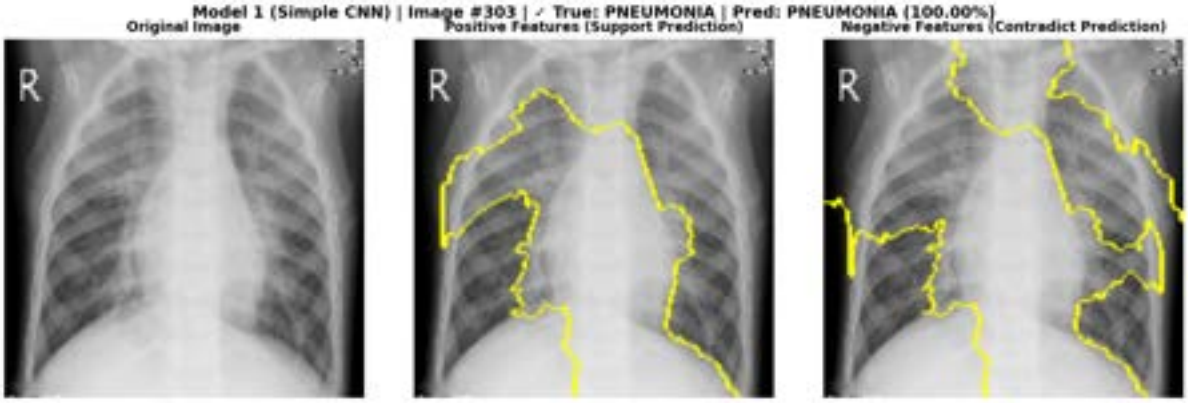

```
Sample 7/10 - Image Index: 430
  Generating LIME explanation for image 430...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



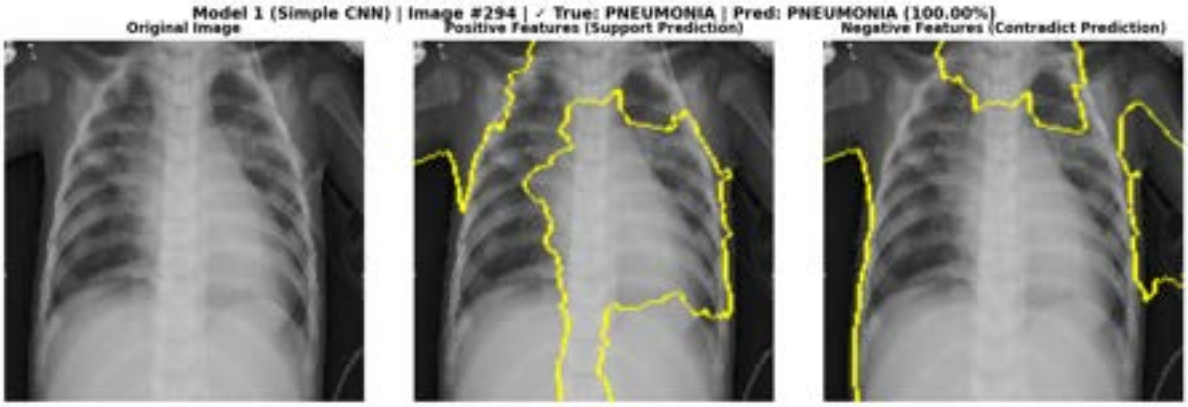

```
Sample 8/10 - Image Index: 387
  Generating LIME explanation for image 387...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



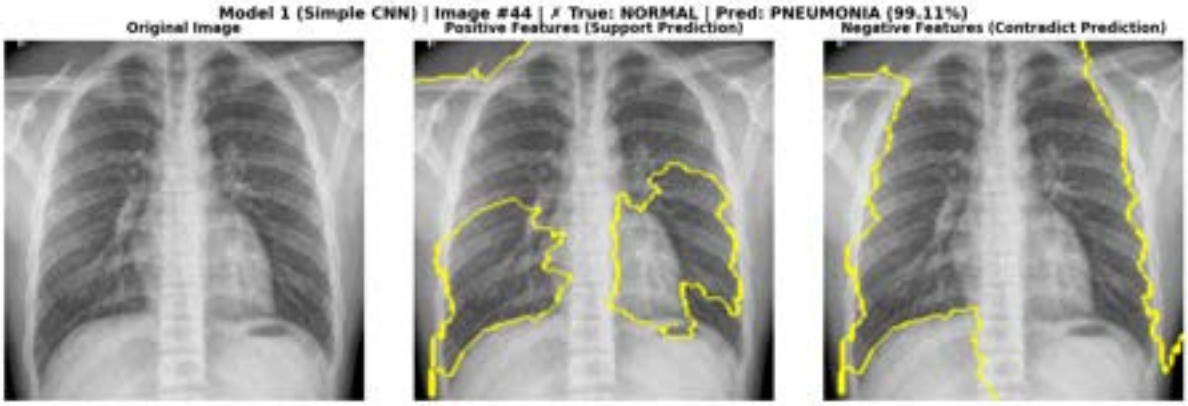

```
Sample 9/10 - Image Index: 303
  Generating LIME explanation for image 303...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



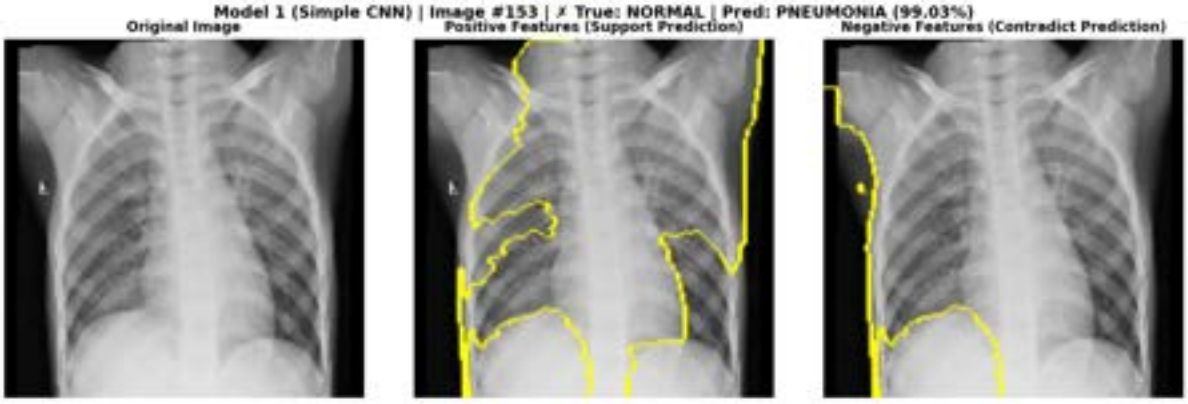

```
Sample 10/10 - Image Index: 294
  Generating LIME explanation for image 294...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



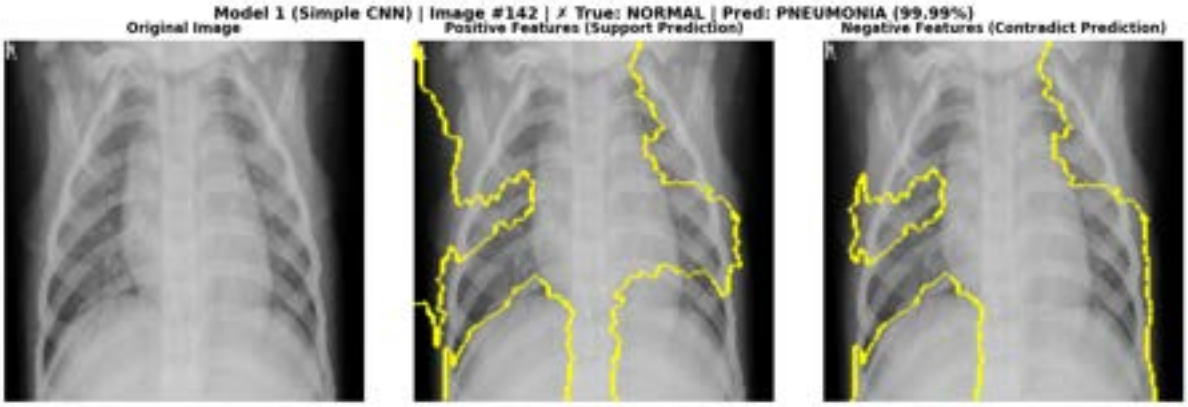

```
============================================================
SCENARIO 3: False Positive (FP) - Normal→Pneumonia ✗
============================================================
Found 119 cases, showing 10 examples...

Sample 1/10 - Image Index: 44
  Generating LIME explanation for image 44...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



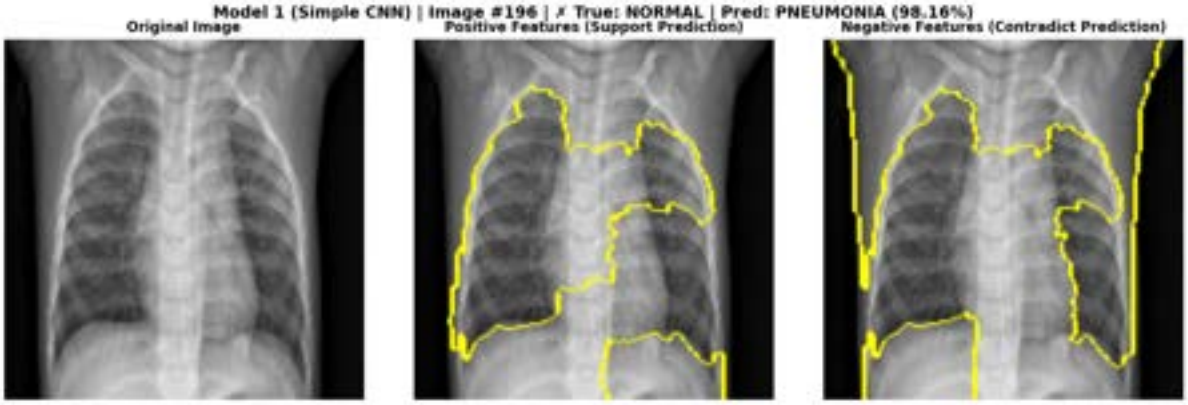

```
Sample 2/10 - Image Index: 153
  Generating LIME explanation for image 153...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



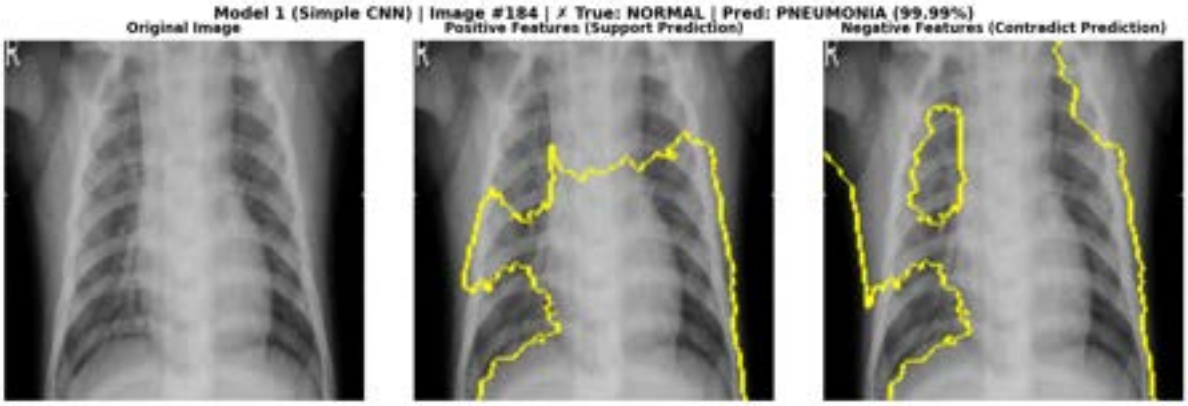

```
Sample 3/10 - Image Index: 142
  Generating LIME explanation for image 142...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



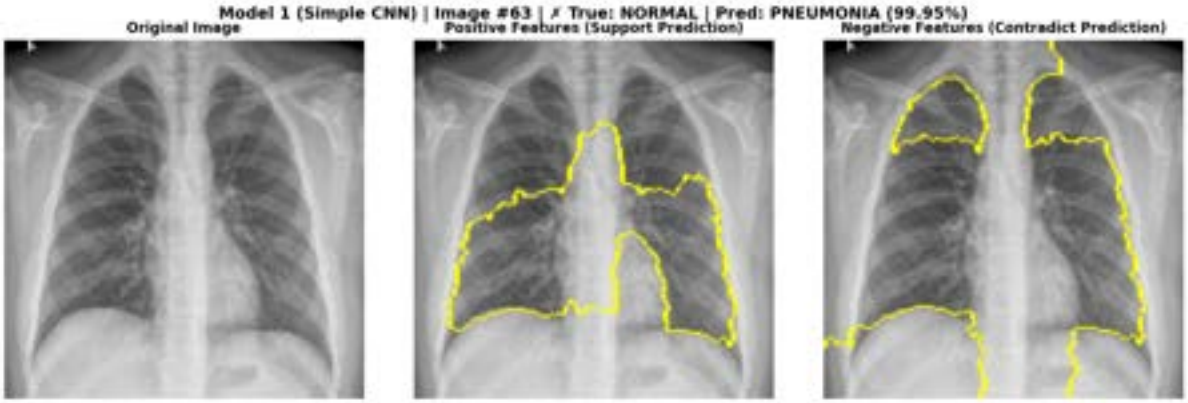

```
Sample 4/10 - Image Index: 196
  Generating LIME explanation for image 196...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



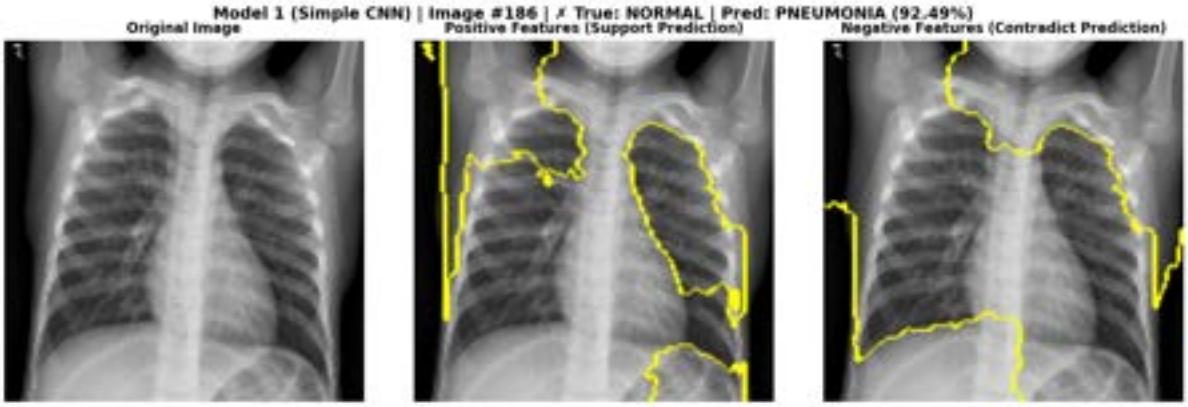
```
Sample 5/10 - Image Index: 184
  Generating LIME explanation for image 184...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



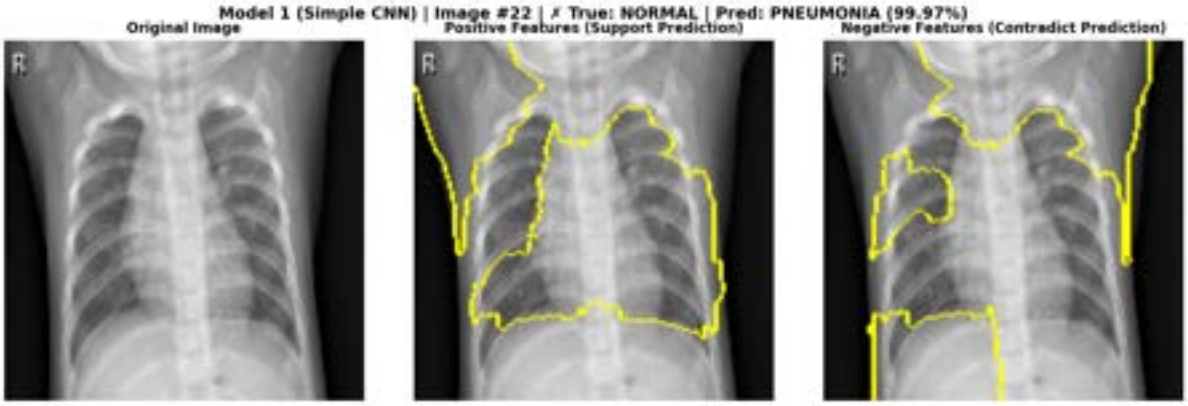

```
Sample 6/10 - Image Index: 63
  Generating LIME explanation for image 63...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



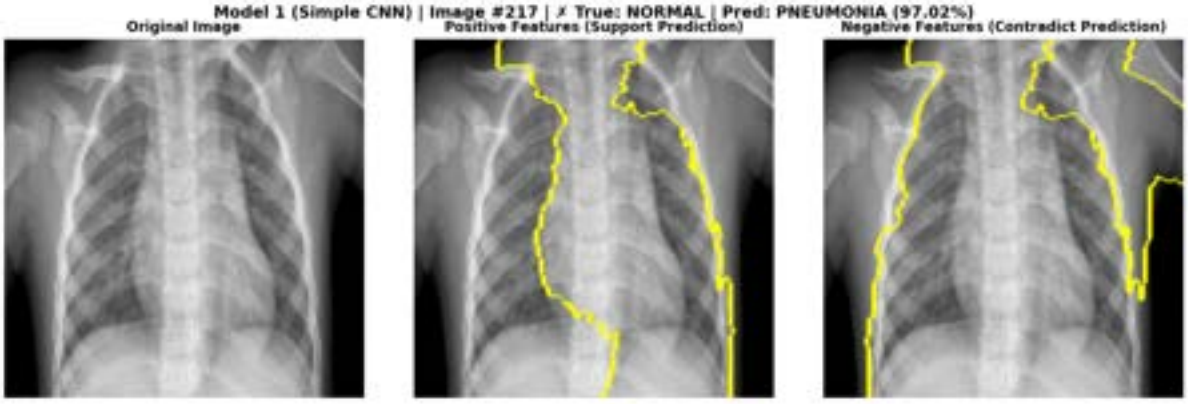

```
Sample 7/10 - Image Index: 186
  Generating LIME explanation for image 186...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



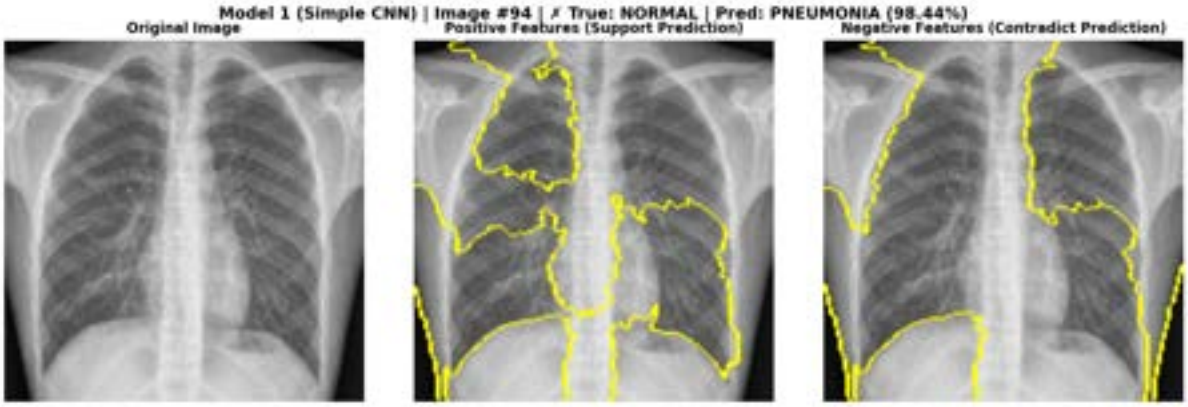

```
Sample 8/10 - Image Index: 22
  Generating LIME explanation for image 22...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



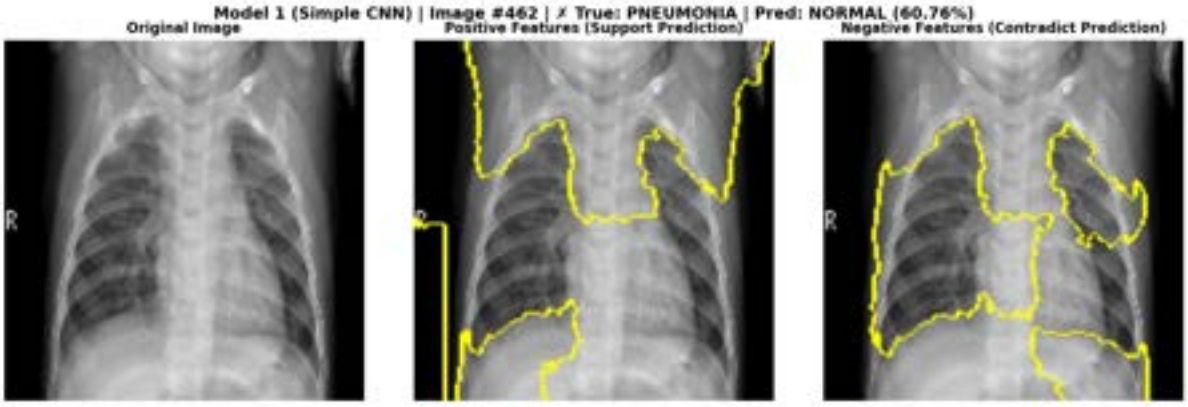

```
Sample 9/10 - Image Index: 217
  Generating LIME explanation for image 217...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



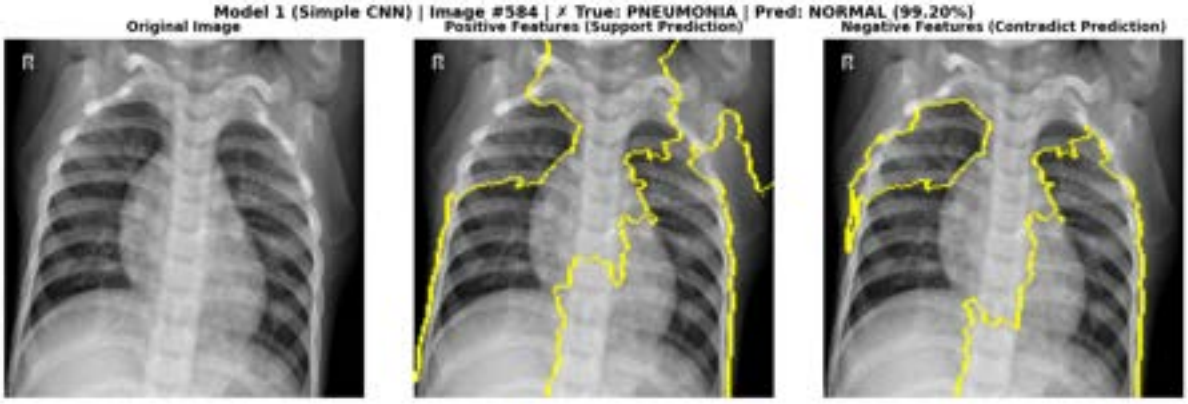

```
Sample 10/10 - Image Index: 94
  Generating LIME explanation for image 94...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



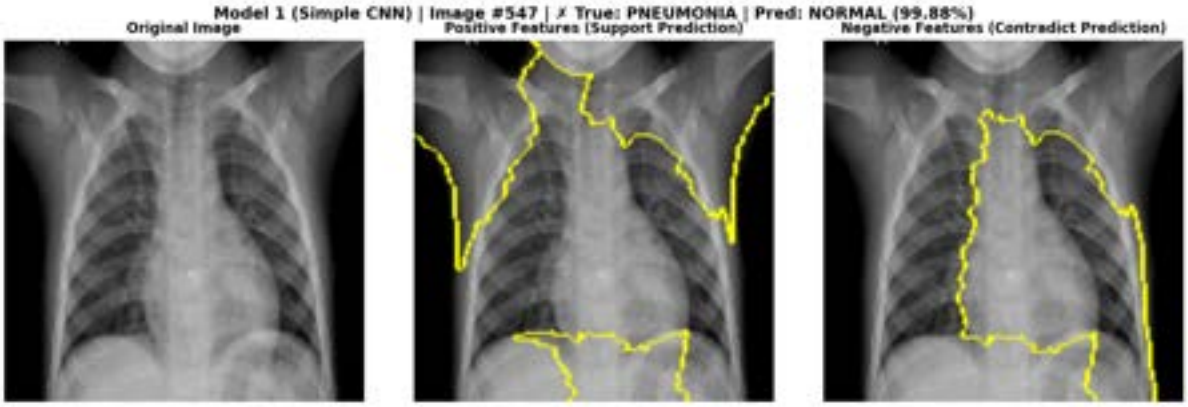

```
============================================================
SCENARIO 4: False Negative (FN) - Pneumonia→Normal ✗
============================================================
Found 3 cases, showing 3 examples...

Sample 1/3 - Image Index: 462
  Generating LIME explanation for image 462...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



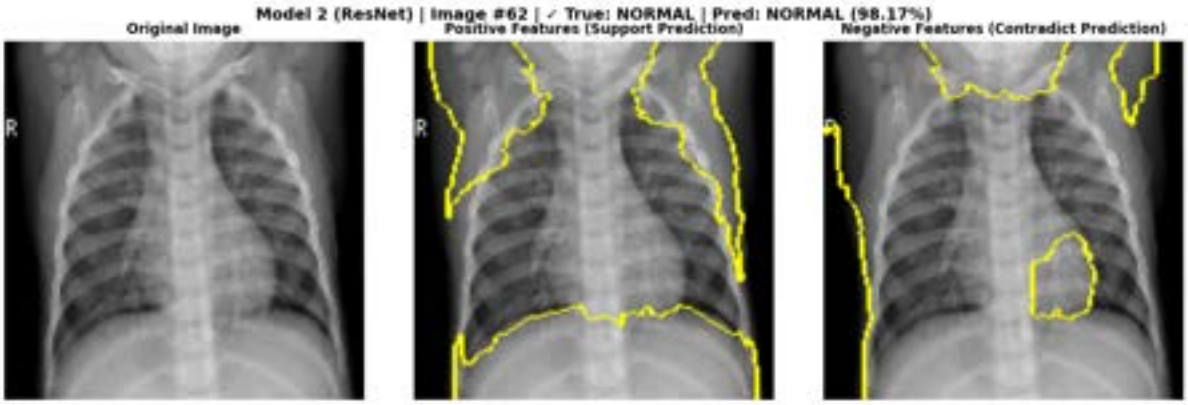

```
Sample 2/3 - Image Index: 584
  Generating LIME explanation for image 584...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```




```
Sample 3/3 - Image Index: 547
  Generating LIME explanation for image 547...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

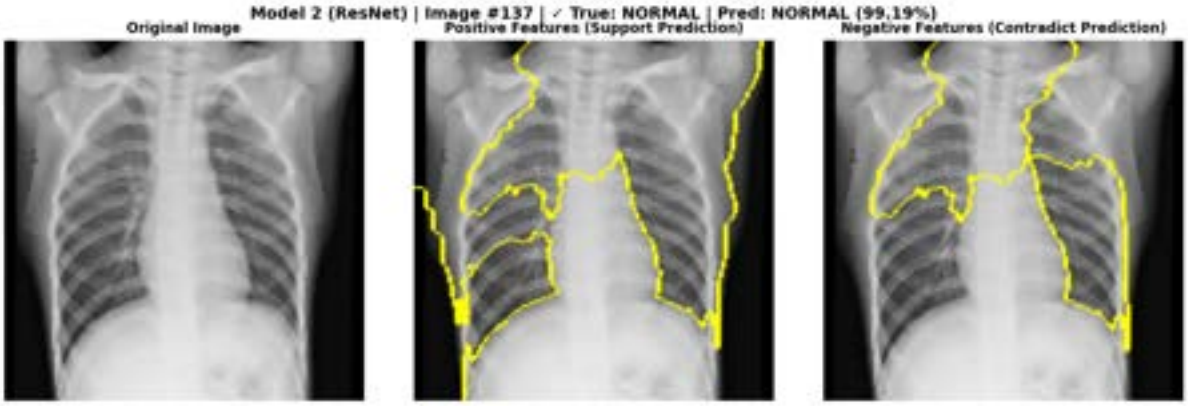


```
============================================================
Model 1 (Simple CNN) - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):   115 (18.43%)
✓ True Positive  (TP):   387 (62.02%)
✗ False Positive (FP):   119 (19.07%)
✗ False Negative (FN):     3 ( 0.48%)
============================================================
Overall Accuracy: 80.45%

============================================================
LIME ANALYSIS: Model 2 (ResNet)
============================================================

============================================================
SCENARIO 1: True Negative (TN) - Normal→Normal ✓
============================================================
Found 105 cases, showing 10 examples...

Sample 1/10 - Image Index: 62
  Generating LIME explanation for image 62...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



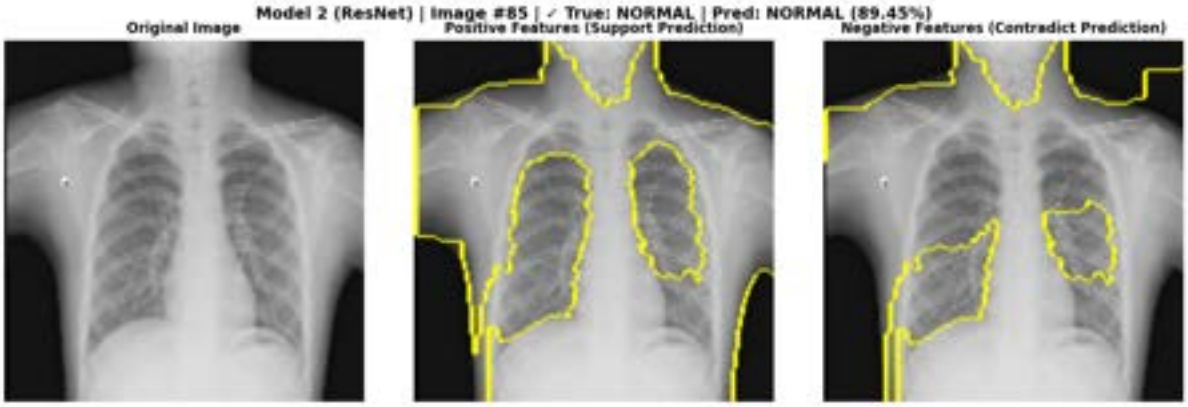

```
Sample 2/10 - Image Index: 140
  Generating LIME explanation for image 140...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```


```
Sample 3/10 - Image Index: 137
  Generating LIME explanation for image 137...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

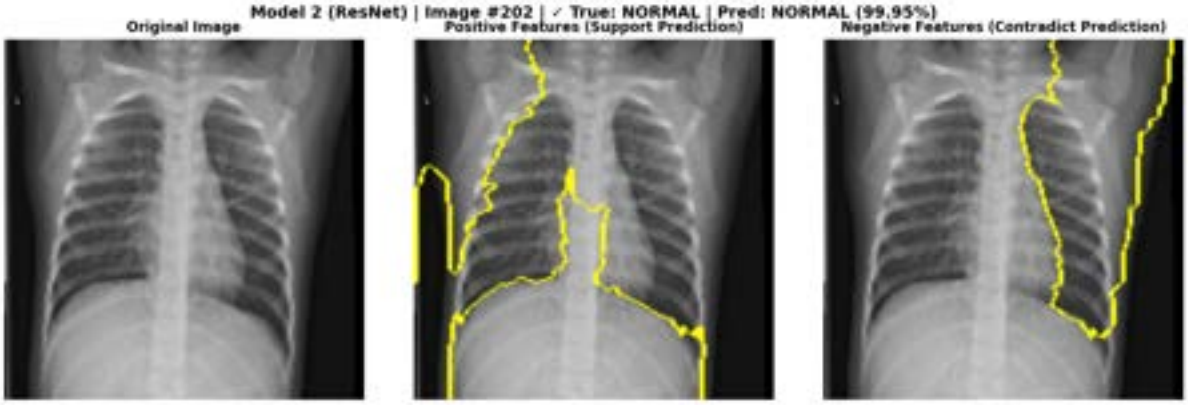



```
Sample 4/10 - Image Index: 116
  Generating LIME explanation for image 116...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

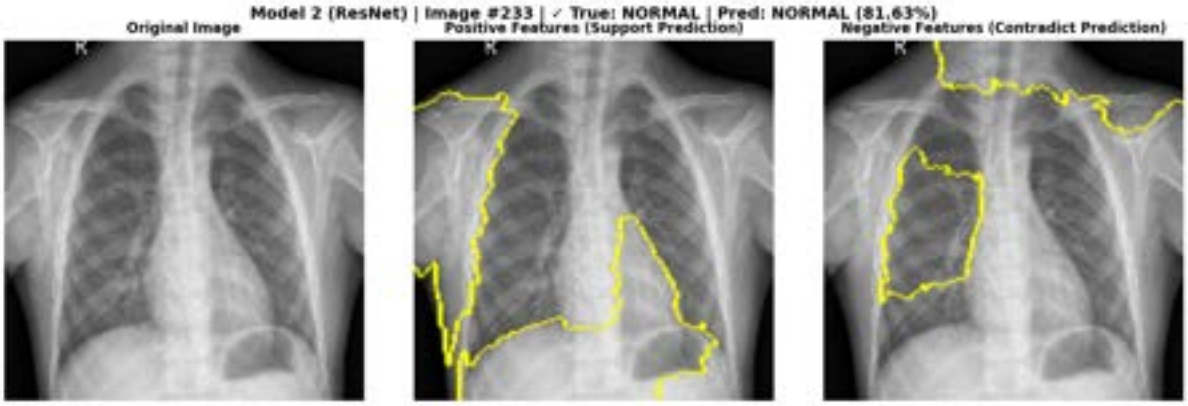

```
Sample 5/10 - Image Index: 85
  Generating LIME explanation for image 85...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



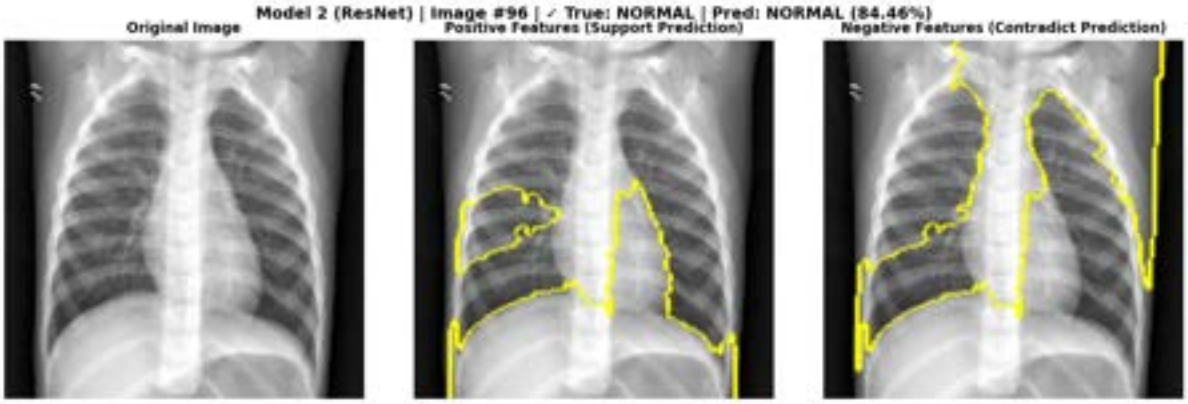

```
Sample 6/10 - Image Index: 202
  Generating LIME explanation for image 202...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



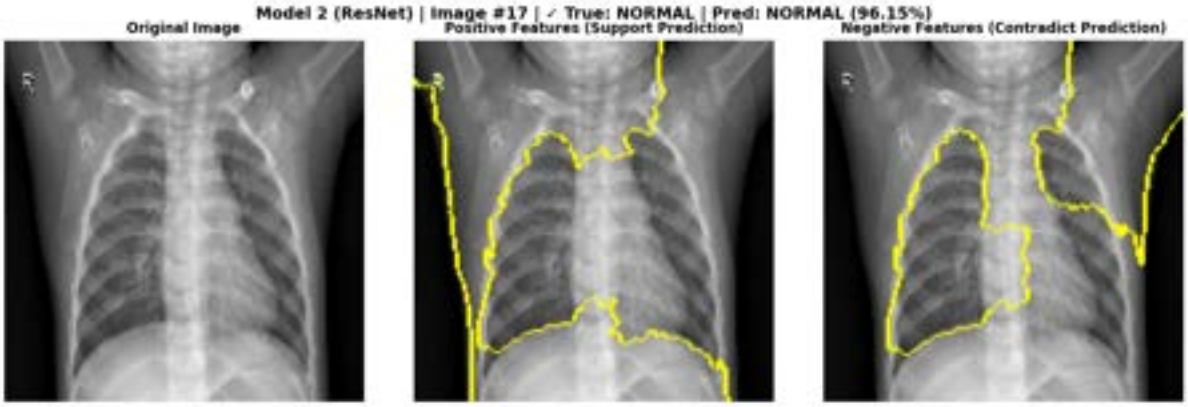
```
Sample 7/10 - Image Index: 233
  Generating LIME explanation for image 233...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



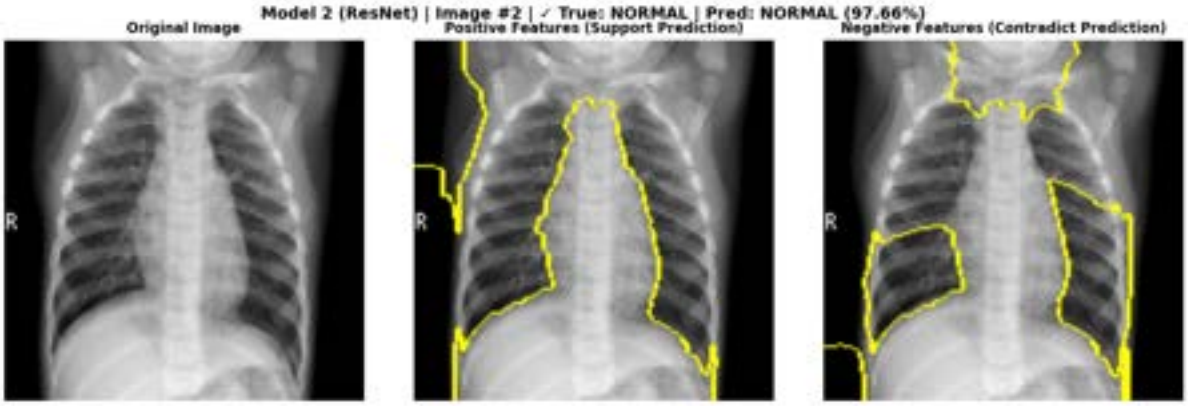

```
Sample 8/10 - Image Index: 96
  Generating LIME explanation for image 96...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



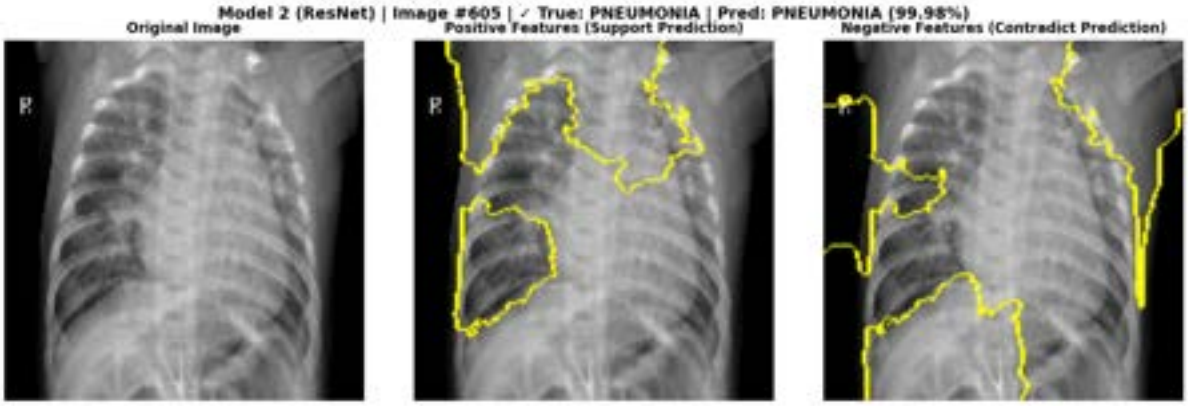

```
Sample 9/10 - Image Index: 17
  Generating LIME explanation for image 17...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



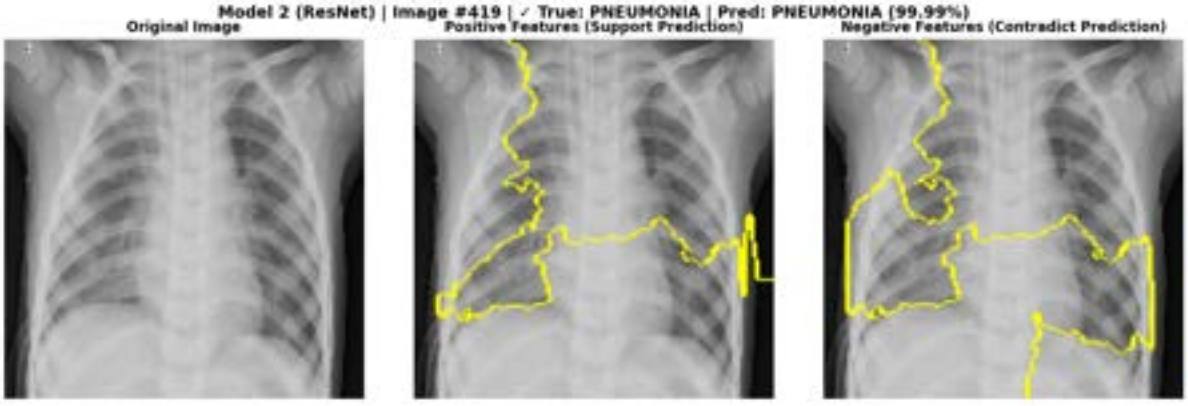

```
Sample 10/10 - Image Index: 2
  Generating LIME explanation for image 2...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



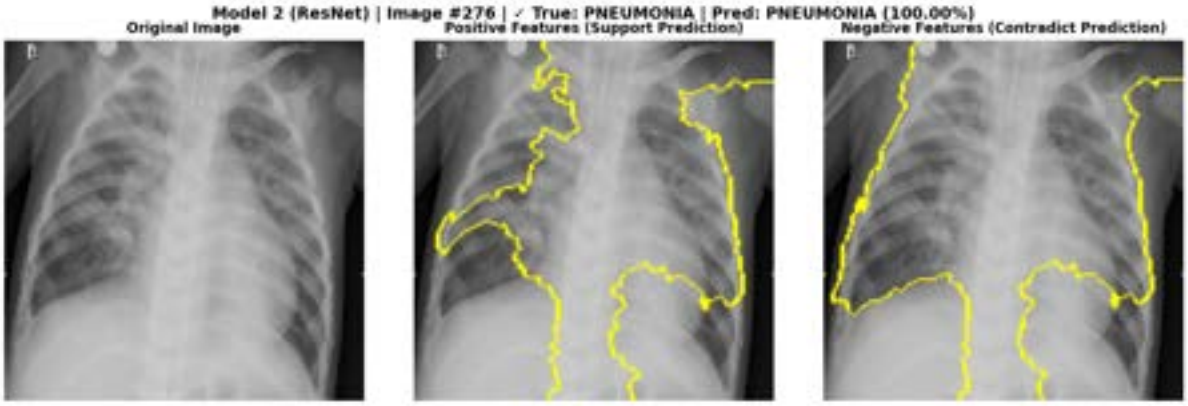

```
============================================================
SCENARIO 2: True Positive (TP) - Pneumonia→Pneumonia ✓
============================================================
Found 386 cases, showing 10 examples...

Sample 1/10 - Image Index: 605
  Generating LIME explanation for image 605...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```




```
Sample 2/10 - Image Index: 419
  Generating LIME explanation for image 419...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

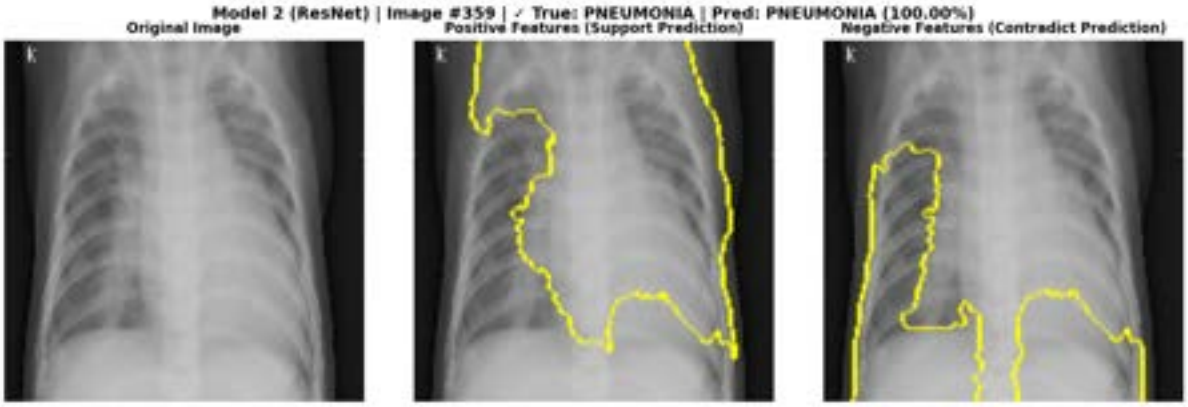



```
Sample 3/10 - Image Index: 276
  Generating LIME explanation for image 276...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

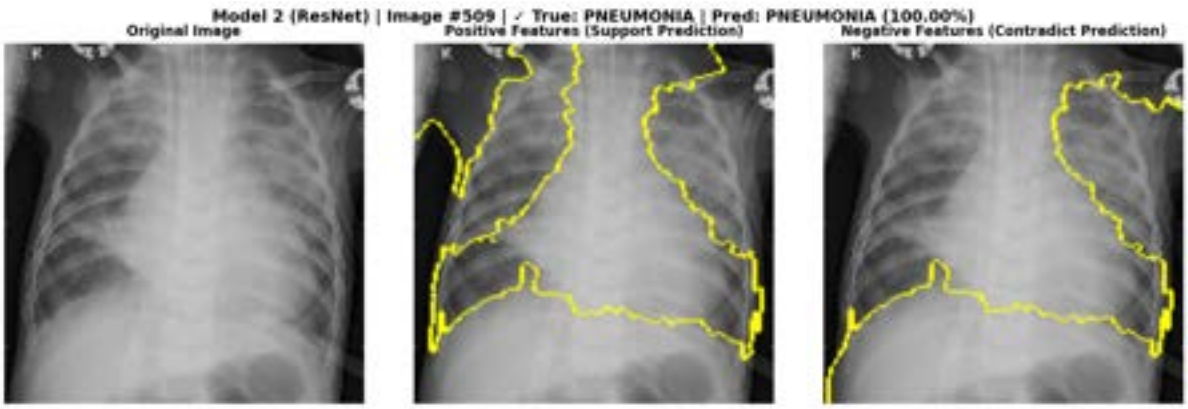



```
Sample 4/10 - Image Index: 258
  Generating LIME explanation for image 258...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```


```
Sample 5/10 - Image Index: 359
  Generating LIME explanation for image 359...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

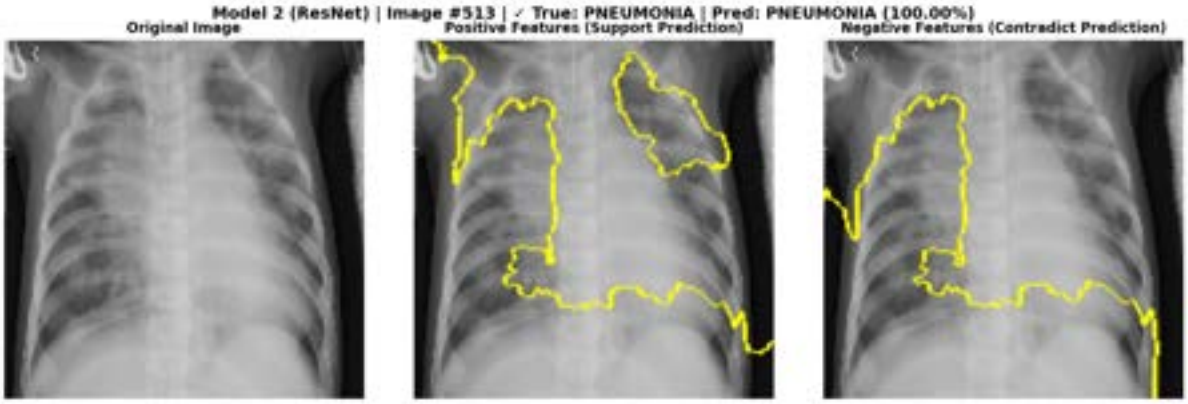



```
Sample 6/10 - Image Index: 509
  Generating LIME explanation for image 509...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

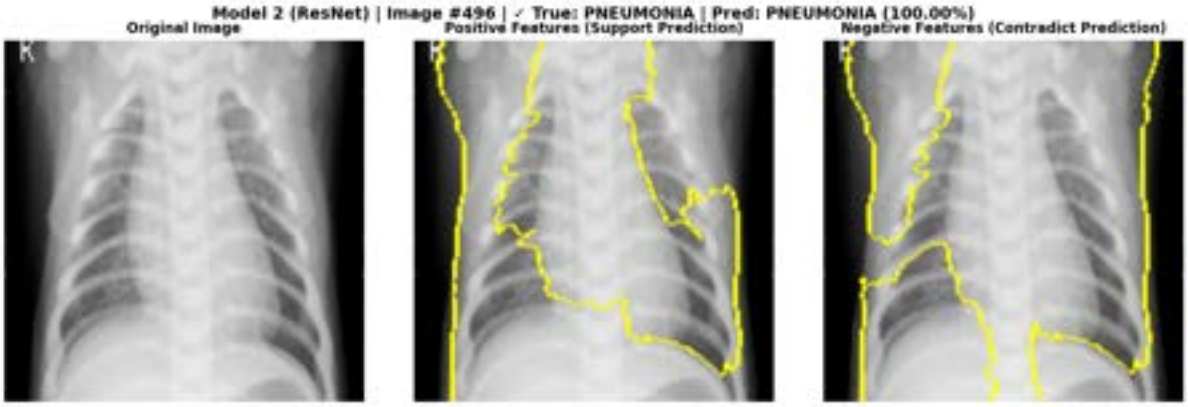


```
Sample 7/10 - Image Index: 263
  Generating LIME explanation for image 263...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

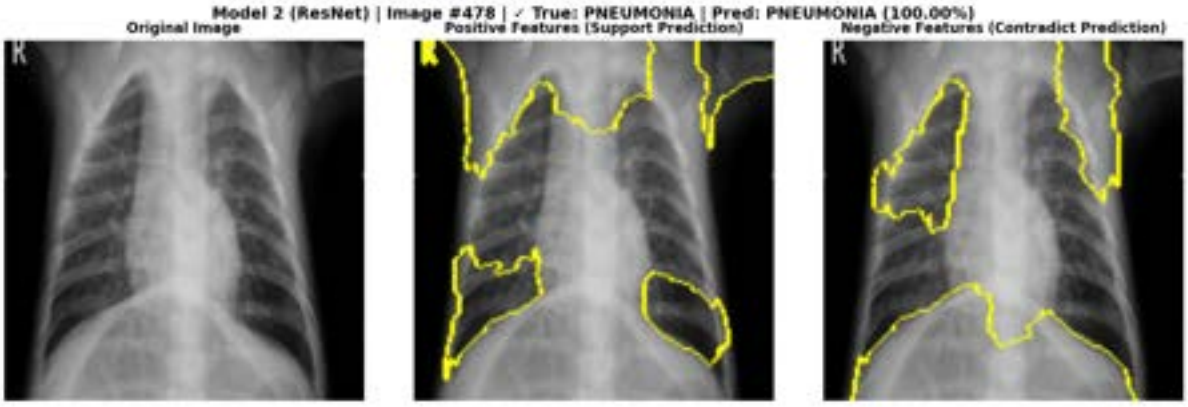

```
Sample 8/10 - Image Index: 513
  Generating LIME explanation for image 513...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```




```
Sample 9/10 - Image Index: 496
  Generating LIME explanation for image 496...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

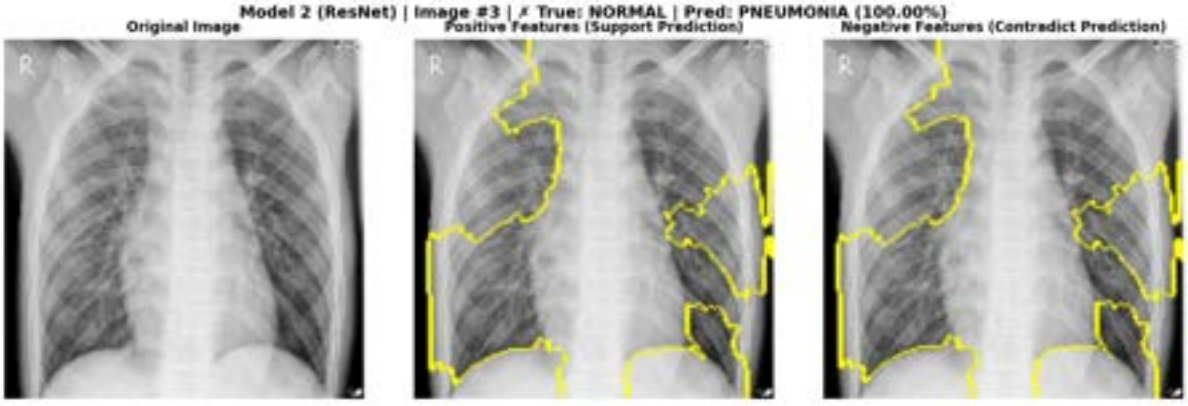



```
Sample 10/10 - Image Index: 478
  Generating LIME explanation for image 478...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

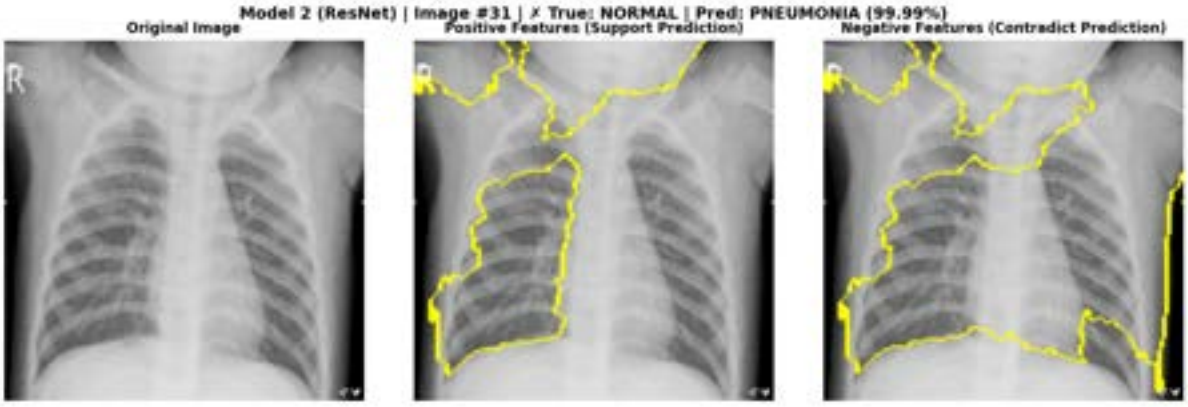



```
============================================================
SCENARIO 3: False Positive (FP) - Normal→Pneumonia ✗
============================================================
Found 129 cases, showing 10 examples...

Sample 1/10 - Image Index: 126
  Generating LIME explanation for image 126...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```


```
Sample 2/10 - Image Index: 3
  Generating LIME explanation for image 3...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

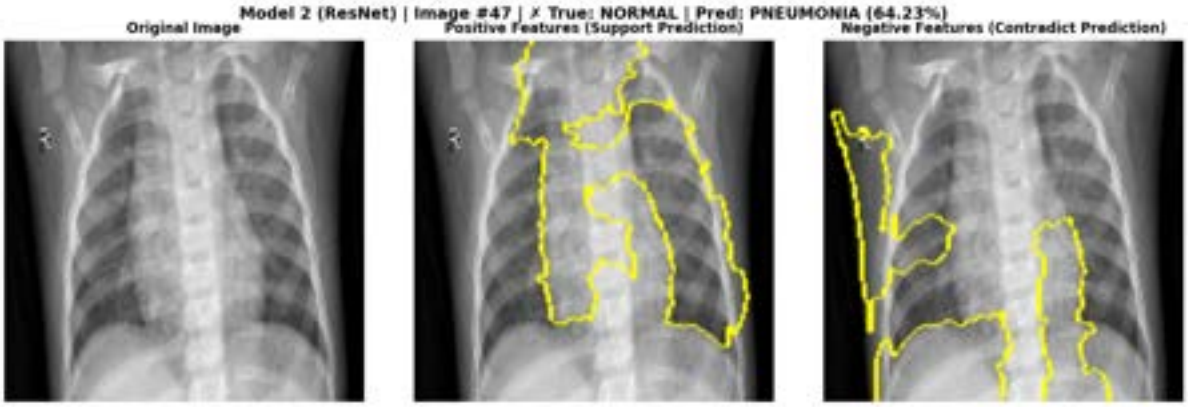



```
Sample 3/10 - Image Index: 31
  Generating LIME explanation for image 31...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

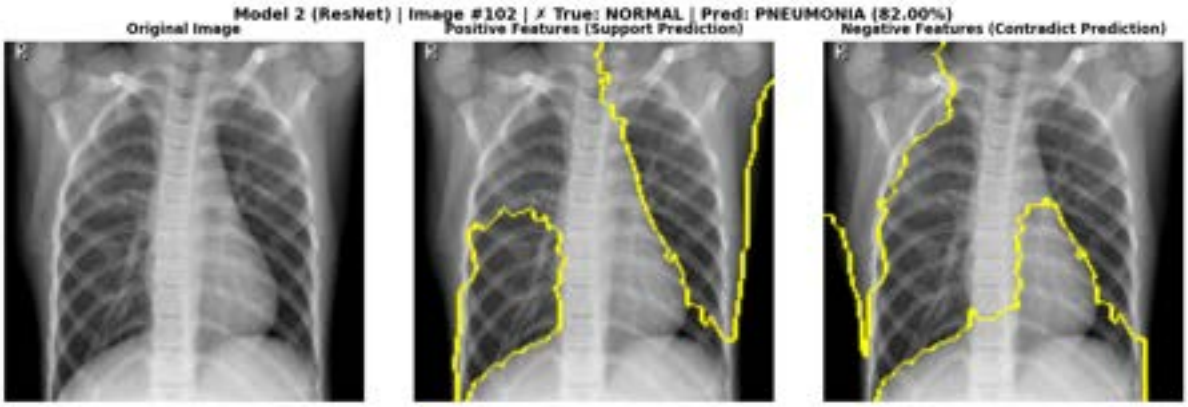



```
Sample 4/10 - Image Index: 46
  Generating LIME explanation for image 46...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

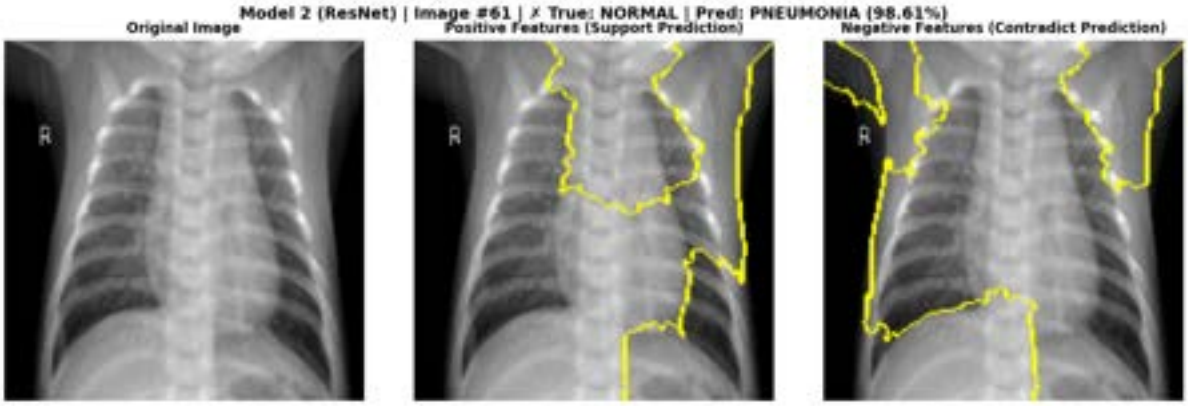

```
Sample 5/10 - Image Index: 47
  Generating LIME explanation for image 47...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



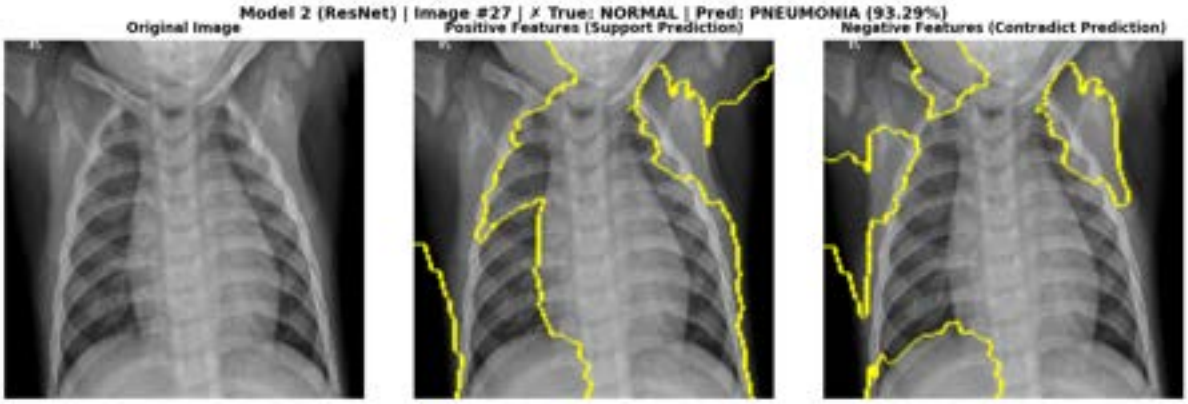

```
Sample 6/10 - Image Index: 102
  Generating LIME explanation for image 102...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



```
Sample 7/10 - Image Index: 61
  Generating LIME explanation for image 61...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

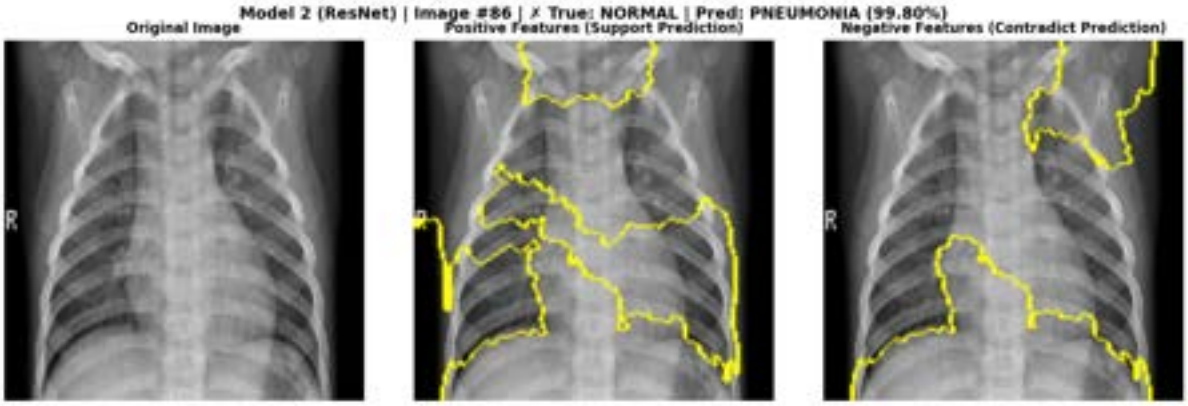



```
Sample 8/10 - Image Index: 27
  Generating LIME explanation for image 27...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

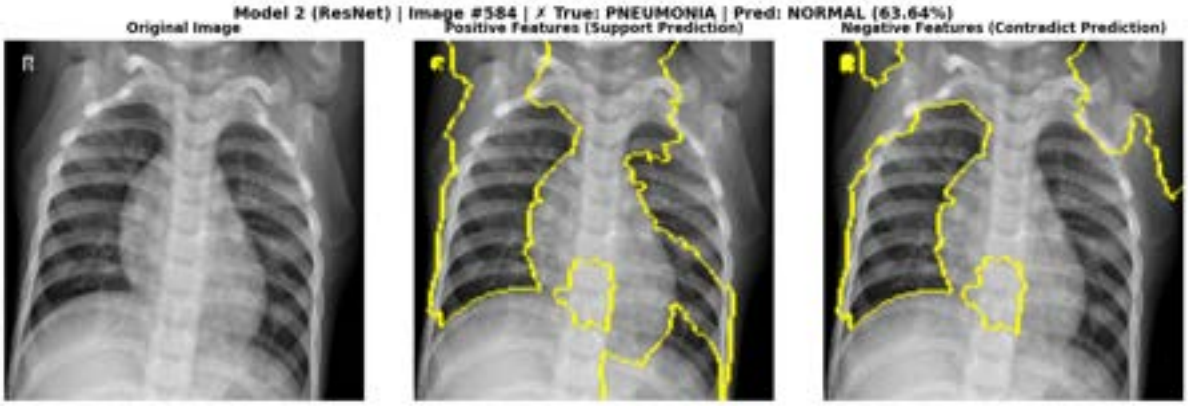



```
Sample 9/10 - Image Index: 229
  Generating LIME explanation for image 229...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

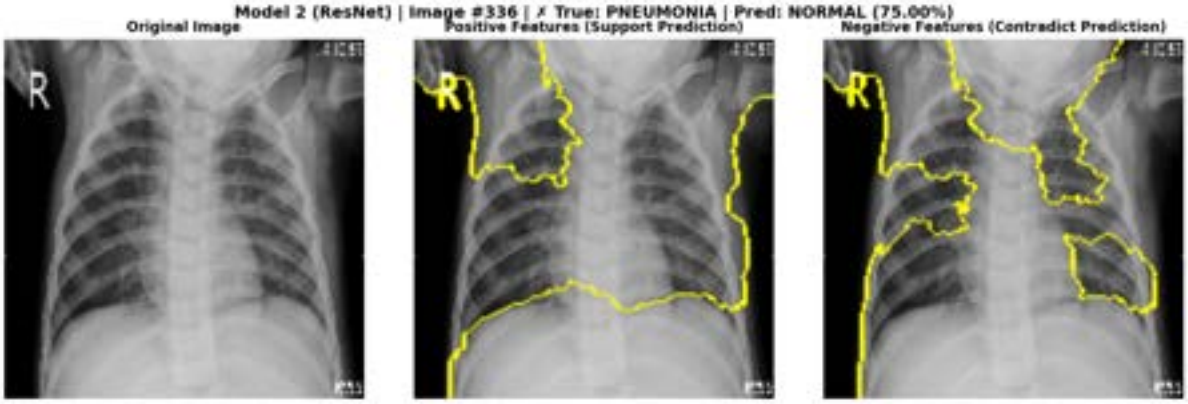

```
Sample 10/10 - Image Index: 86
  Generating LIME explanation for image 86...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



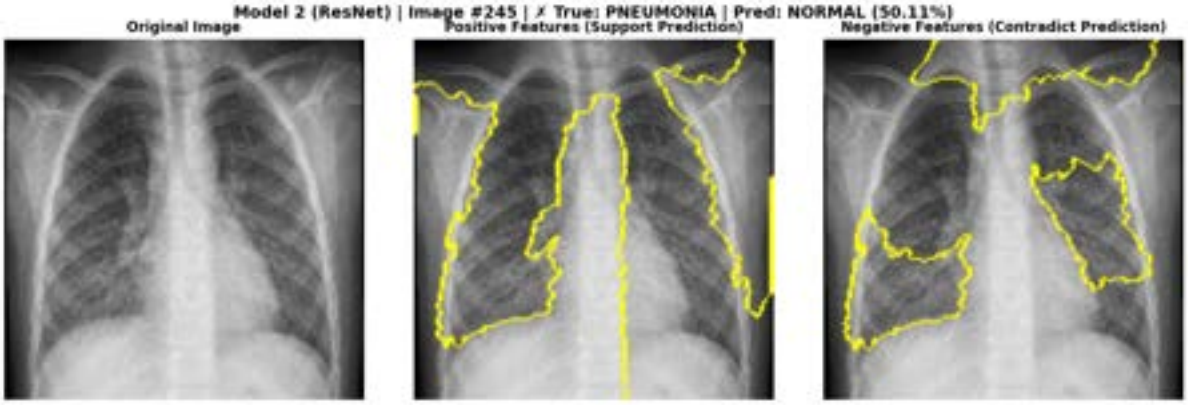

```
============================================================
SCENARIO 4: False Negative (FN) - Pneumonia→Normal ✗
============================================================
Found 4 cases, showing 4 examples...

Sample 1/4 - Image Index: 584
  Generating LIME explanation for image 584...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```



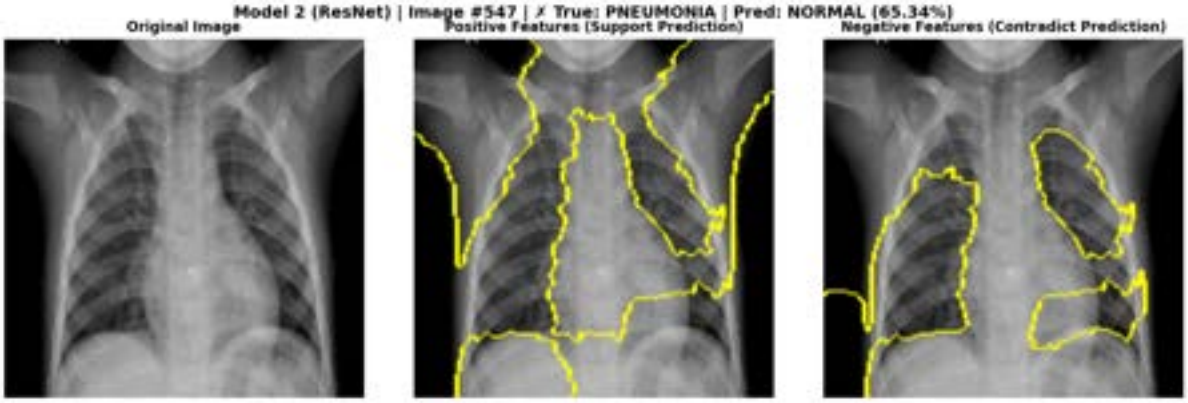

```
Sample 2/4 - Image Index: 336
  Generating LIME explanation for image 336...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```




```
Sample 3/4 - Image Index: 245
  Generating LIME explanation for image 245...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```




```
Sample 4/4 - Image Index: 547
  Generating LIME explanation for image 547...
  0%|          | 0/1000 [00:00<?, ?it/s]
Done!
```

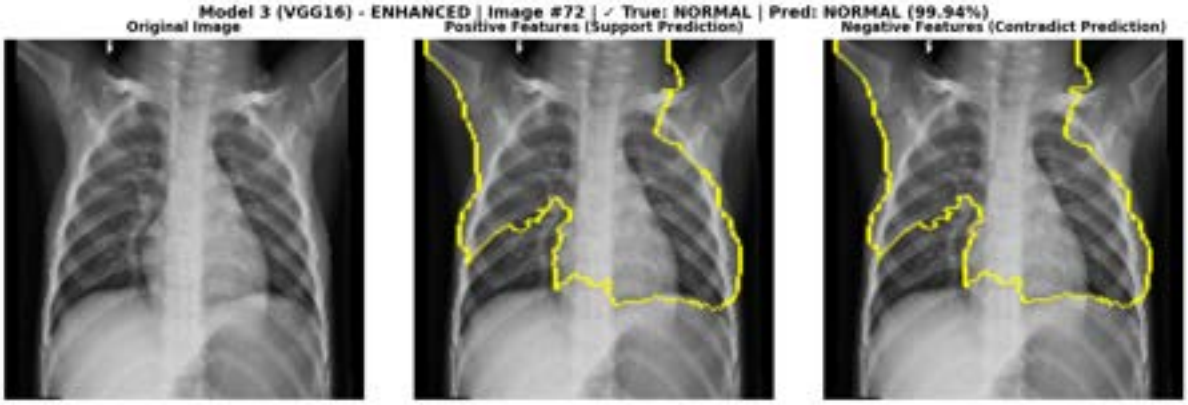



```
============================================================
Model 2 (ResNet) - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):   105 (16.83%)
✓ True Positive  (TP):   386 (61.86%)
✗ False Positive (FP):   129 (20.67%)
✗ False Negative (FN):     4 ( 0.64%)
============================================================
Overall Accuracy: 78.69%

============================================================
LIME ANALYSIS: Model 3 (VGG16) - ENHANCED
============================================================

============================================================
SCENARIO 1: True Negative (TN) - Normal→Normal ✓
============================================================
Found 81 cases, showing 10 examples...

Sample 1/10 - Image Index: 71
  Generating LIME explanation for image 71...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```

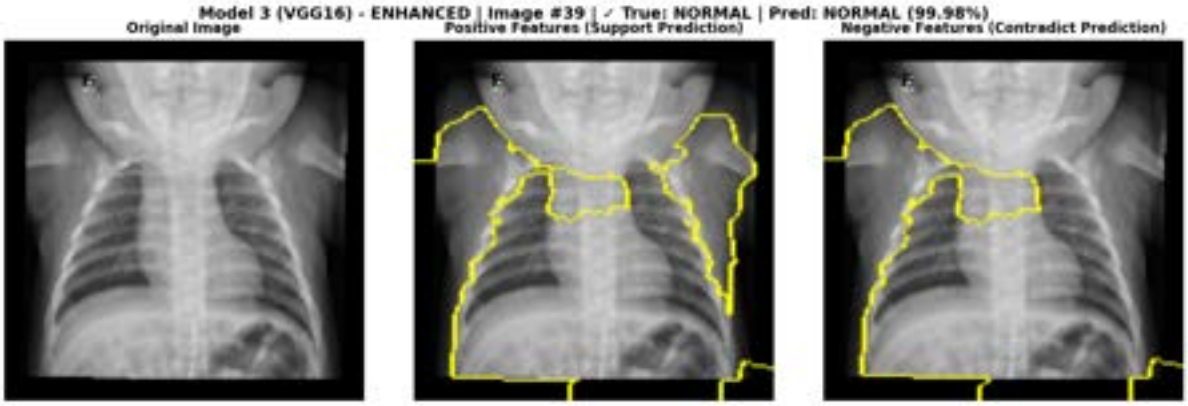

```
Sample 2/10 - Image Index: 2
  Generating LIME explanation for image 2...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```


```
Sample 3/10 - Image Index: 54
  Generating LIME explanation for image 54...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```


```
Sample 4/10 - Image Index: 72
  Generating LIME explanation for image 72...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



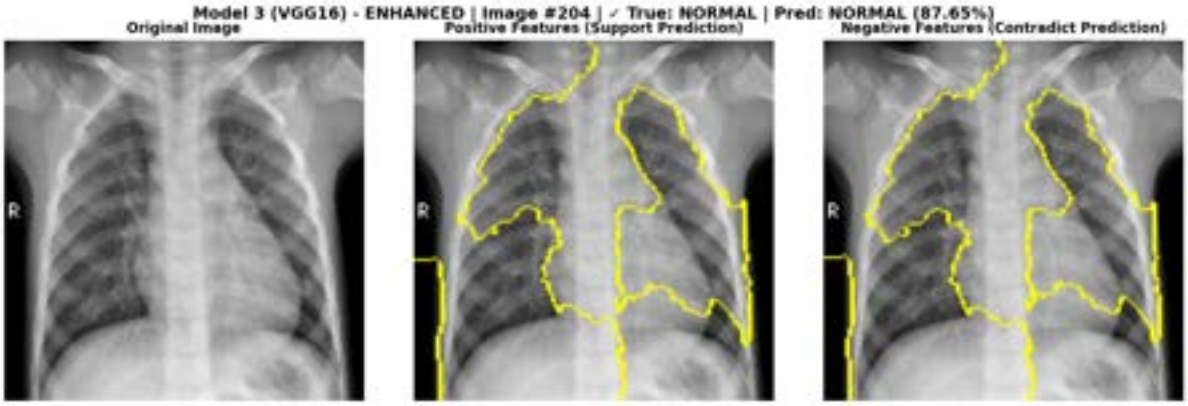
```
Sample 5/10 - Image Index: 39
  Generating LIME explanation for image 39...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```




```
Sample 6/10 - Image Index: 64
  Generating LIME explanation for image 64...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```


```
Sample 7/10 - Image Index: 19
  Generating LIME explanation for image 19...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```

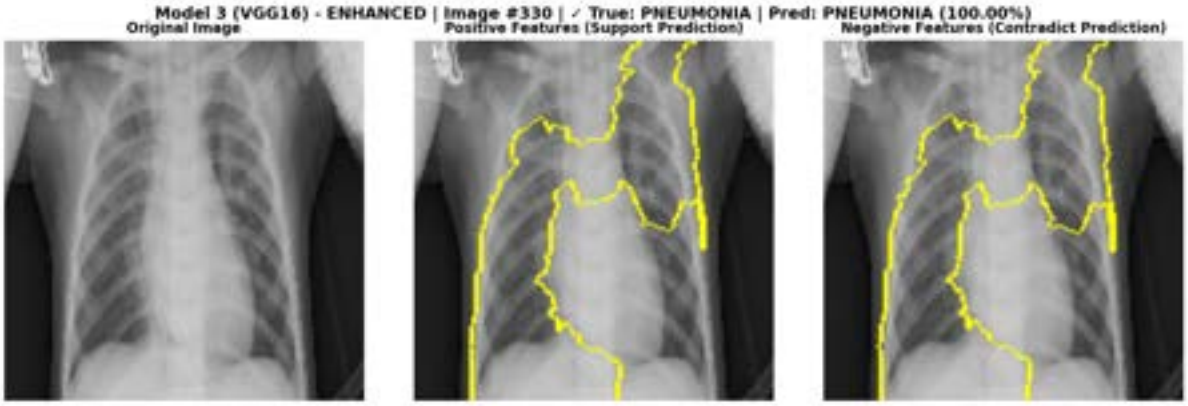

```
Sample 8/10 - Image Index: 204
  Generating LIME explanation for image 204...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



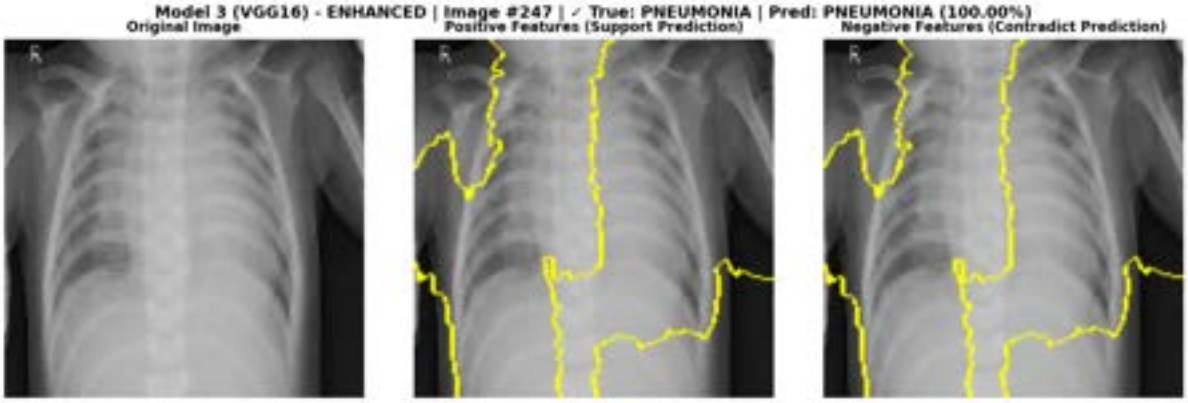

```
Sample 9/10 - Image Index: 10
  Generating LIME explanation for image 10...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```


```
Sample 10/10 - Image Index: 23
  Generating LIME explanation for image 23...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```

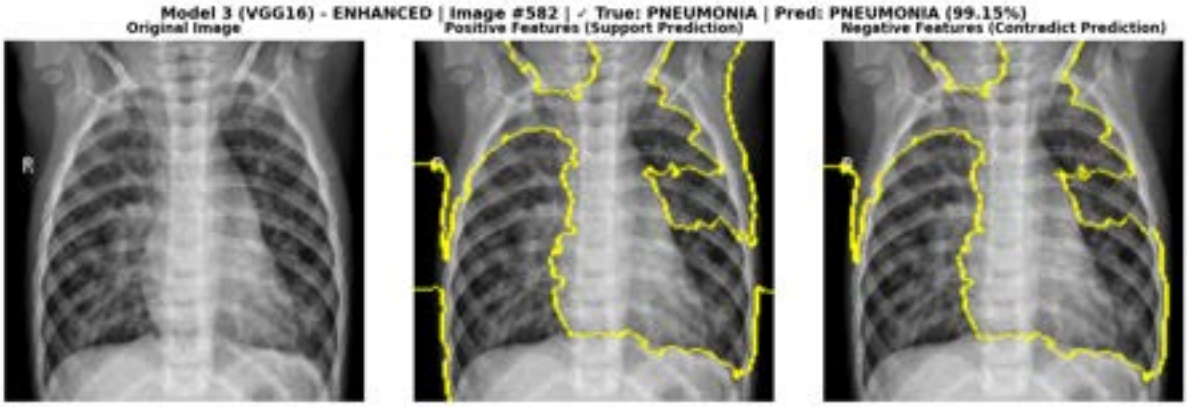

```
============================================================
SCENARIO 2: True Positive (TP) - Pneumonia→Pneumonia ✓
============================================================
Found 388 cases, showing 10 examples...

Sample 1/10 - Image Index: 330
  Generating LIME explanation for image 330...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



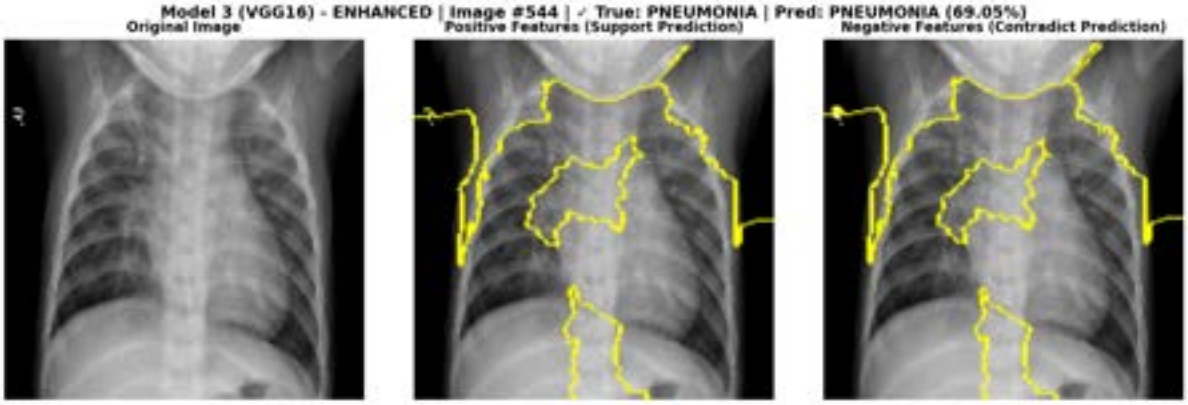

```
Sample 2/10 - Image Index: 247
  Generating LIME explanation for image 247...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



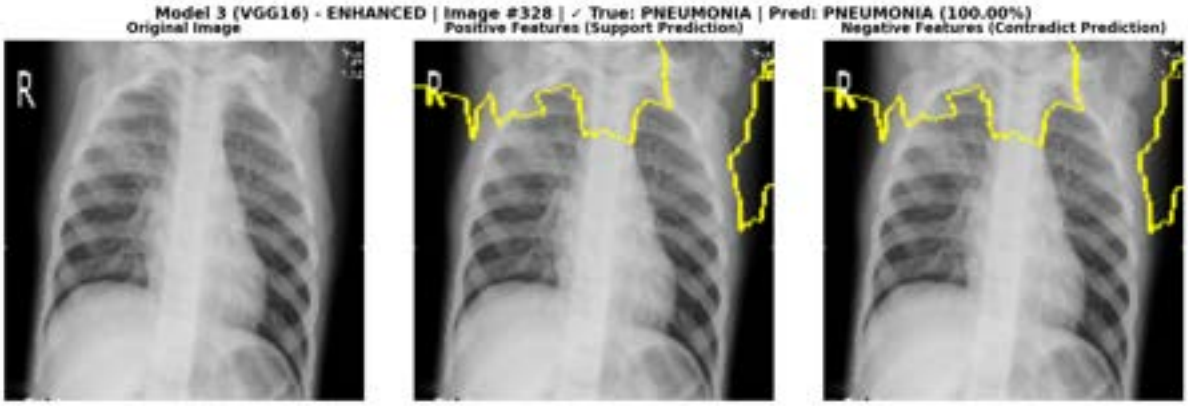

```
Sample 3/10 - Image Index: 582
  Generating LIME explanation for image 582...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



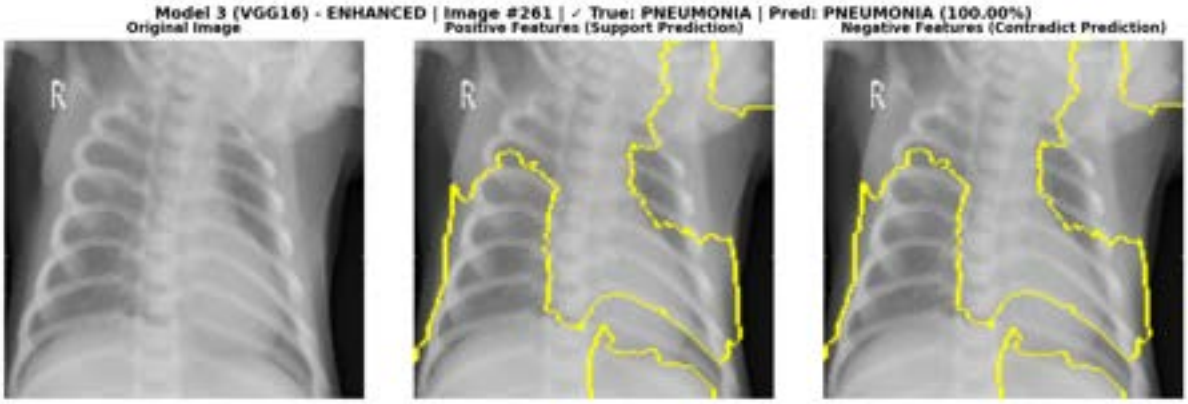

```
Sample 4/10 - Image Index: 544
  Generating LIME explanation for image 544...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



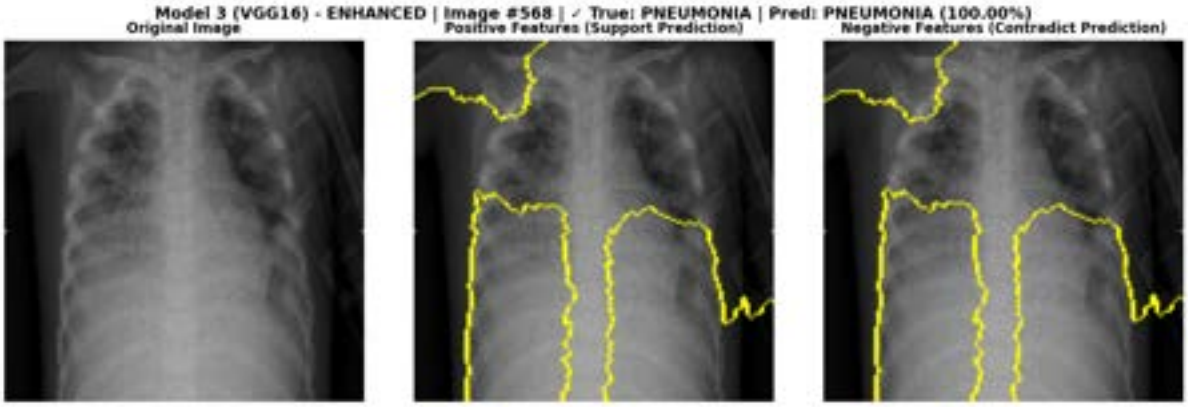
```
Sample 5/10 - Image Index: 328
  Generating LIME explanation for image 328...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



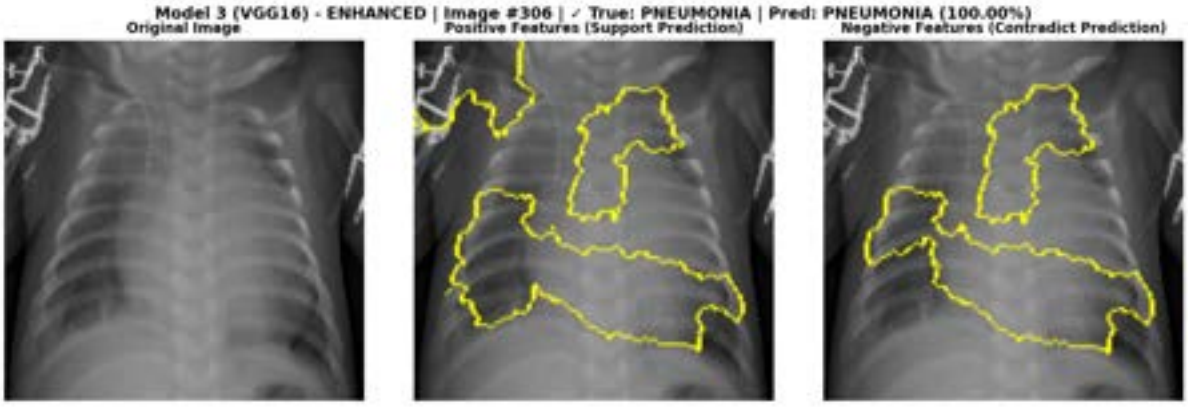

```
Sample 6/10 - Image Index: 261
  Generating LIME explanation for image 261...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



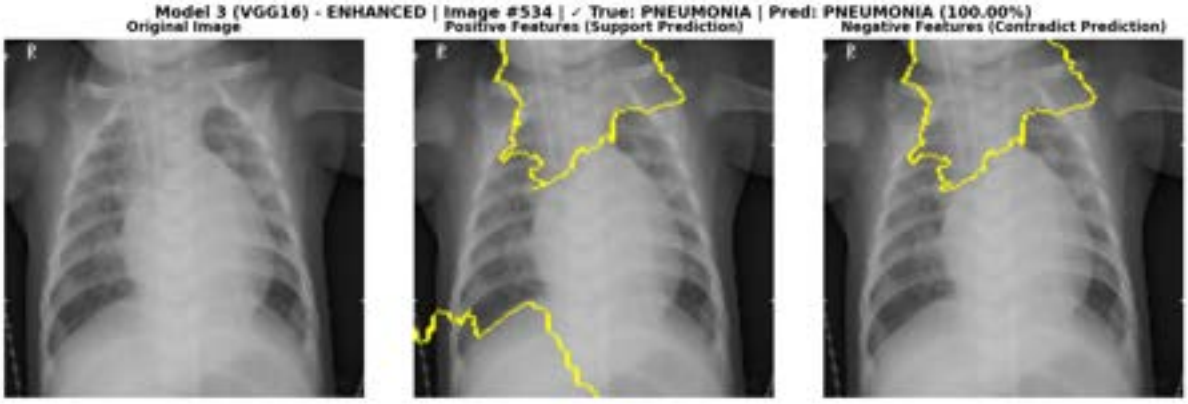

```
Sample 7/10 - Image Index: 568
  Generating LIME explanation for image 568...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



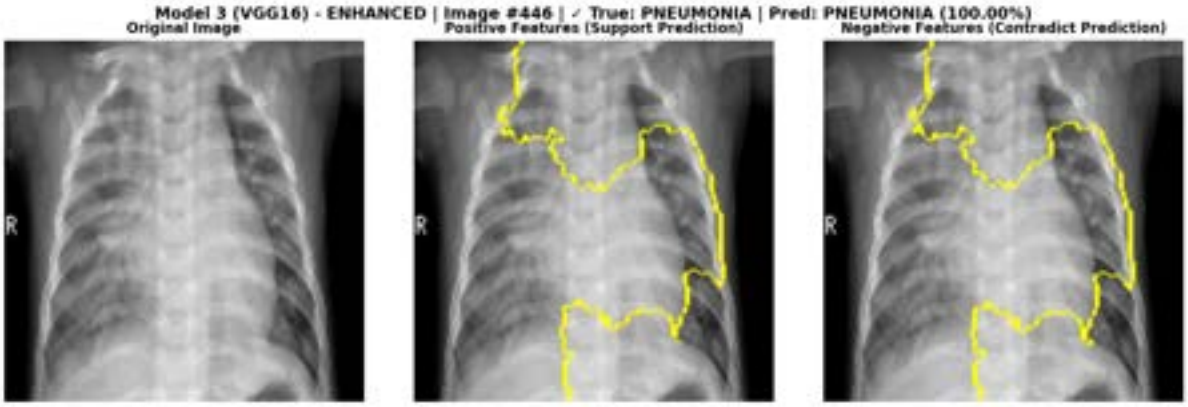

```
Sample 8/10 - Image Index: 306
  Generating LIME explanation for image 306...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



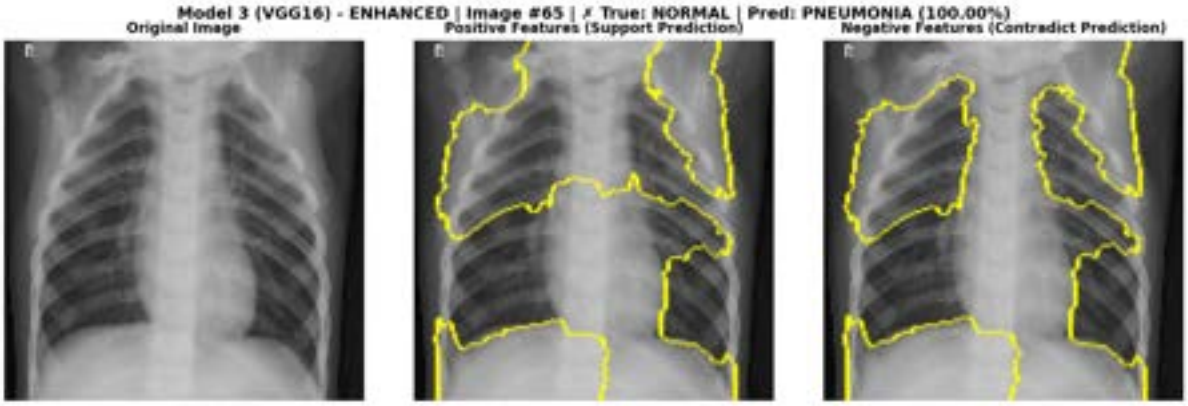

```
Sample 9/10 - Image Index: 534
  Generating LIME explanation for image 534...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



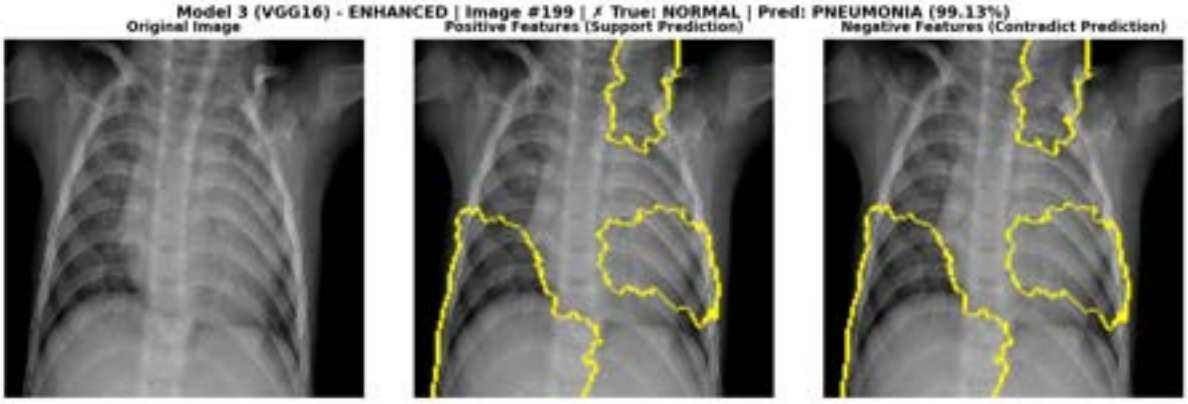

```
Sample 10/10 - Image Index: 446
  Generating LIME explanation for image 446...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



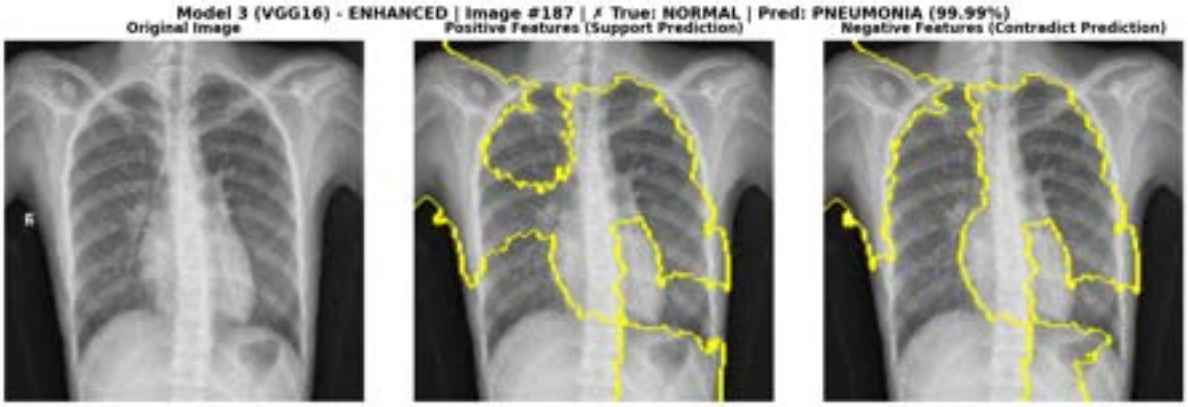

```
============================================================
SCENARIO 3: False Positive (FP) - Normal→Pneumonia ✗
============================================================
Found 153 cases, showing 10 examples...

Sample 1/10 - Image Index: 65
  Generating LIME explanation for image 65...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



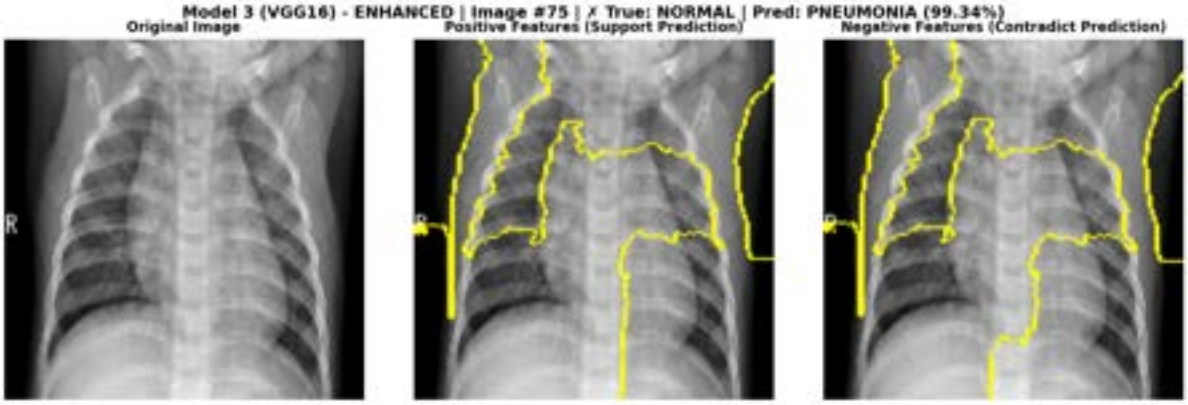

```
Sample 2/10 - Image Index: 199
  Generating LIME explanation for image 199...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



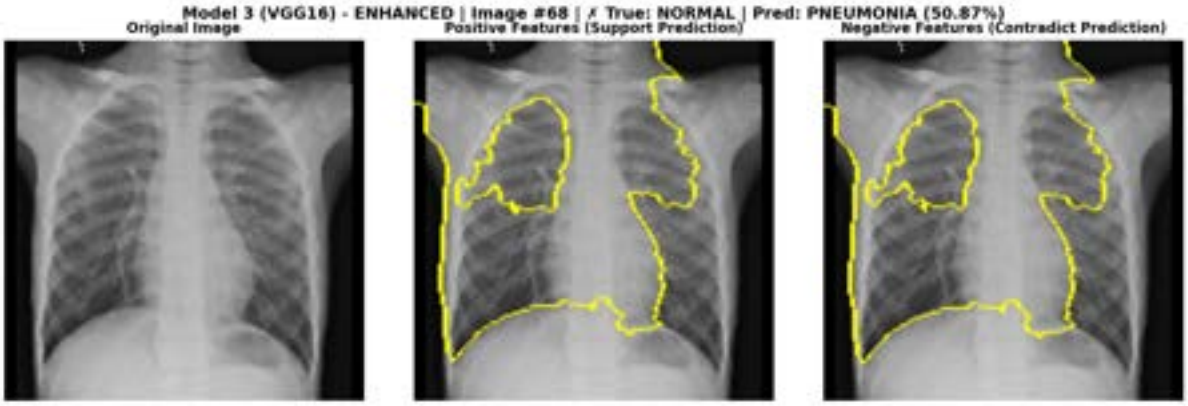

```
Sample 3/10 - Image Index: 187
  Generating LIME explanation for image 187...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



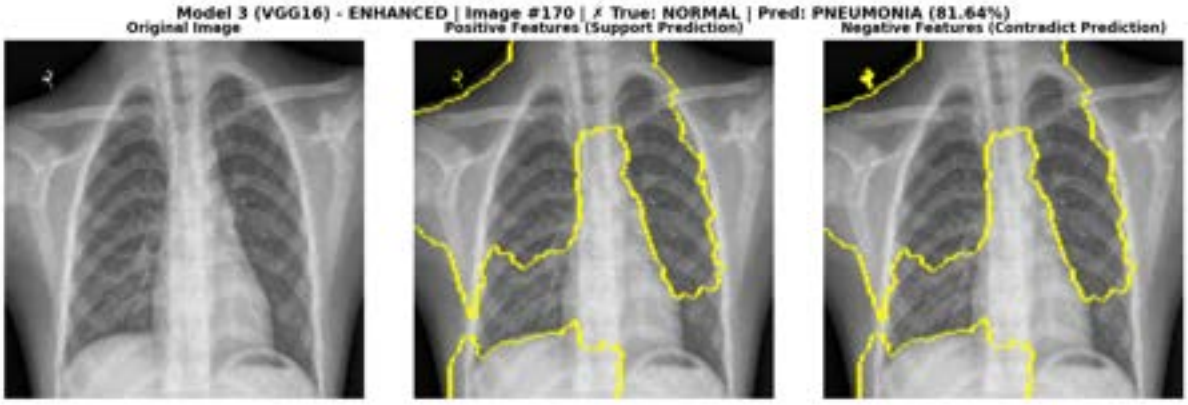

```
Sample 4/10 - Image Index: 75
  Generating LIME explanation for image 75...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



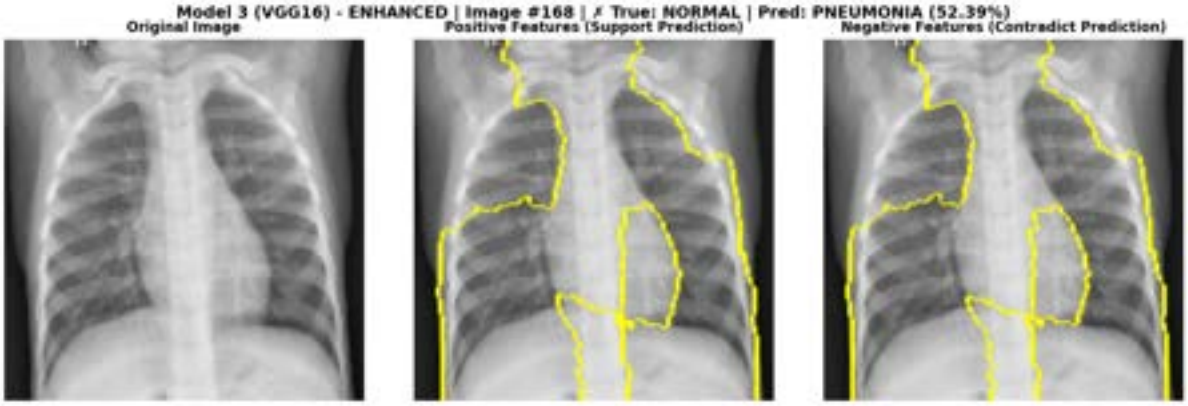
```
Sample 5/10 - Image Index: 68
  Generating LIME explanation for image 68...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



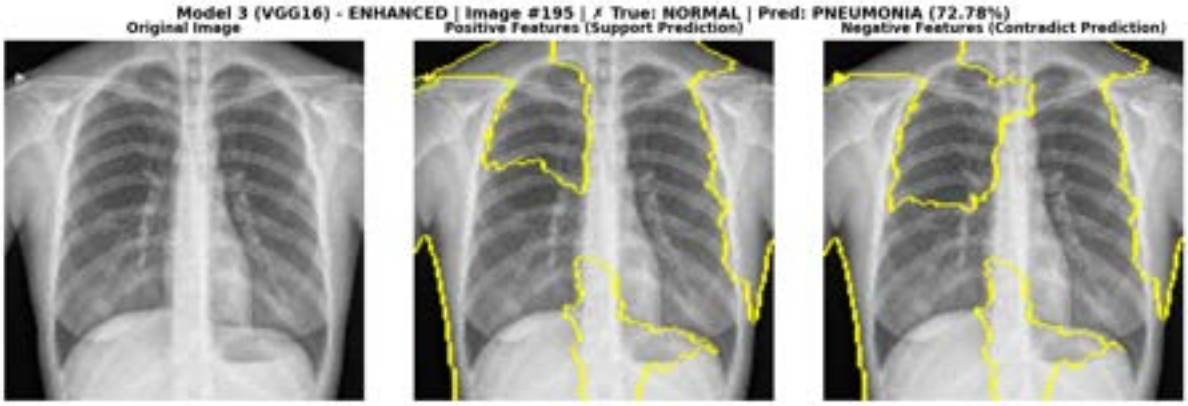

```
Sample 6/10 - Image Index: 170
  Generating LIME explanation for image 170...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



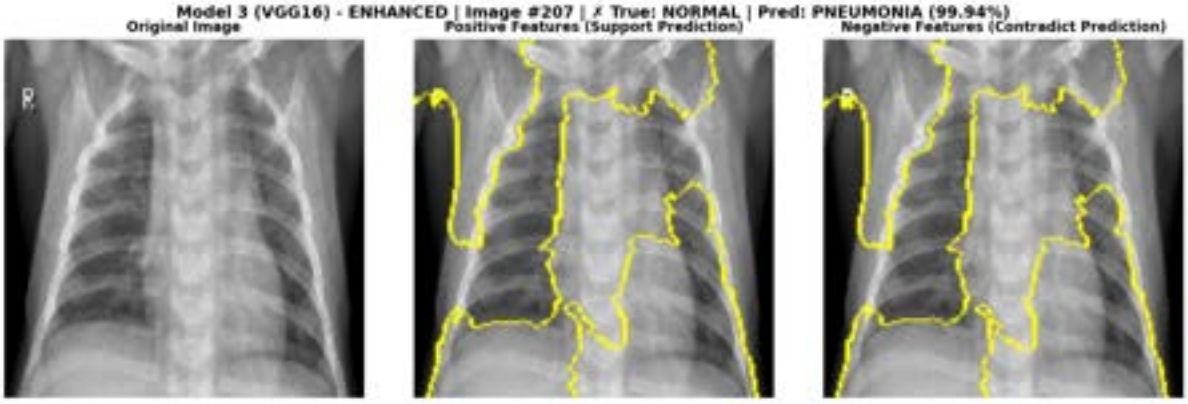

```
Sample 7/10 - Image Index: 168
  Generating LIME explanation for image 168...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



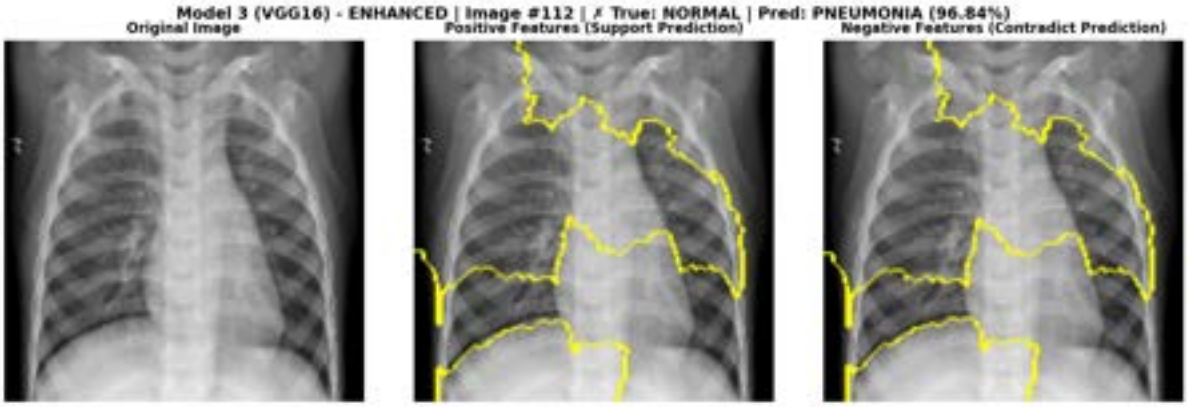

```
Sample 8/10 - Image Index: 195
  Generating LIME explanation for image 195...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



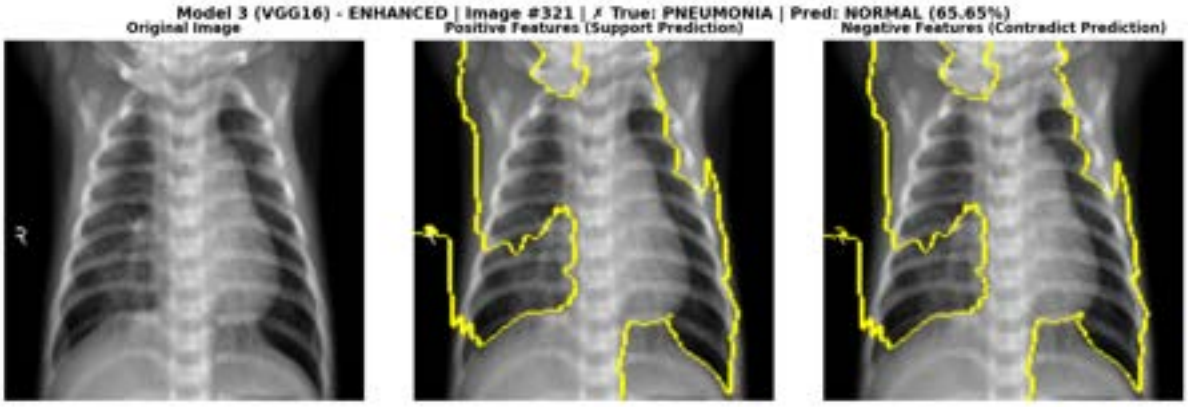

```
Sample 9/10 - Image Index: 207
  Generating LIME explanation for image 207...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```



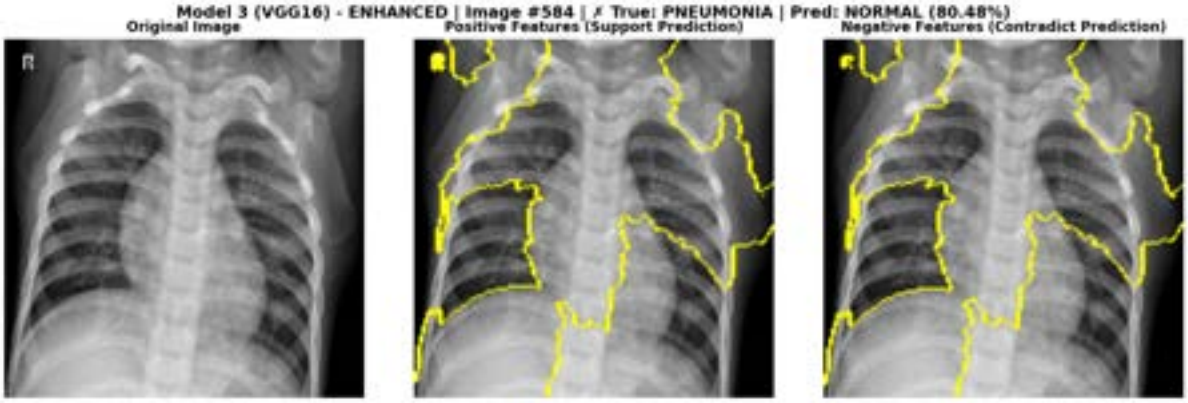

```
Sample 10/10 - Image Index: 112
  Generating LIME explanation for image 112...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```




```
============================================================
SCENARIO 4: False Negative (FN) - Pneumonia→Normal ✗
============================================================
Found 2 cases, showing 2 examples...

Sample 1/2 - Image Index: 321
  Generating LIME explanation for image 321...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```




```
Sample 2/2 - Image Index: 584
  Generating LIME explanation for image 584...
  0%|          | 0/2000 [00:00<?, ?it/s]
Done!
```




```
============================================================
Model 3 (VGG16) - ENHANCED - PREDICTION SUMMARY
============================================================

Test Set Breakdown (Total: 624 samples):
============================================================
✓ True Negative  (TN):    81 (12.98%)
✓ True Positive  (TP):   388 (62.18%)
✗ False Positive (FP):   153 (24.52%)
✗ False Negative (FN):     2 ( 0.32%)
============================================================
Overall Accuracy: 75.16%

============================================================
LIME ANALYSIS COMPLETE!
============================================================

MODEL 3 ENHANCEMENTS:
  ✓ More Samples (2000 vs 1000) - Better accuracy
  ✓ More Features (10 vs 5) - More detailed explanations
  ✓ Superpixel Segmentation - Region-based analysis
  ✓ Positive/Negative Feature Separation - Clear interpretation
============================================================
```

In [ ]: